NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COMPUTER PROGRAMS SUPPORTING INSTRUCTION IN ACOUSTICS

by

Thomas Alan Green

December, 1996

Thesis Advisor:	James V. Sanders
Co- Advisor:	Anthony A. Atchley

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE COMPUTER PROGRAMS SUPPORTING INSTRUCTION IN ACOUSTICS	5. FUNDING NUMBERS	
6. AUTHOR(S) Thomas A. Green		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE	

13. ABSTRACT *(maximum 200 words)*

Traditionally, the study of mechanical vibration and sound wave propagation has been presented through textbooks, classroom discussion and laboratory experiments. However, in today's academic environment, students have access to high performance computing facilities which can greatly augment the learning process. This thesis provides computer algorithms for examining selected topics drawn from the text, Fundamentals of Acoustics, Third Edition, John Wiley & Sons, Inc., by Kinsler, Frey, Coppens and Sanders, (KFCS). Emphasis is on using the modeling and simulation capability of the programing language, MATLAB^{TM}, to illustrate and analyze complex physical principles which may seem obscure on the printed page yet are challenging or inconvenient to duplicate in the laboratory. This is not a passive recitation of acoustic phenomena, but complements KFCS with interactive student participation. The usefulness of these programs and any weaknesses in format or content needs to be tested in the classroom.

14. SUBJECT TERMS acoustics, MATLAB, computer, vibration			15. NUMBER OF PAGES240
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

COMPUTER PROGRAMS SUPPORTING INSTRUCTION IN ACOUSTICS

Thomas A. Green
Lieutenant Commander, United States Navy
B.S., U.S. Naval Academy, 1981

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING ACOUSTICS

from the

NAVAL POSTGRADUATE SCHOOL
December 1996

Author: Thomas A. Green
Thomas A. Green

Approved by: [illegible]
James V. Sanders, Thesis Advisor

[illegible]
Anthony A. Atchley, Co-Advisor

Robert Keolian
Robert M. Keolian, Chairman
Engineering Acoustics Academic Group

ABSTRACT

Traditionally, the study of mechanical vibration and sound wave propagation has been presented through textbooks, classroom discussion and laboratory experiments. However, in today's academic environment, students have access to high performance computing facilities which can greatly augment the learning process. This thesis provides computer algorithms for examining selected topics drawn from the text, Fundamentals of Acoustics, Third Edition, John Wiley & Sons, Inc., by Coppens, Frey, Kinsler and Sanders, (KFCS). Emphasis is on using the modeling and simulation capability of the programing language, MATLAB^{TM}, to illustrate and analyze complex physical principles which may seem obscure on the printed page yet are challenging or inconvenient to duplicate in the laboratory. This is not a passive recitation of acoustic phenomena, but complements KFCS with interactive student participation. The usefulness of these programs and any weaknesses in format or content needs to be tested in the classroom.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Traditionally, the study of mechanical vibration and sound wave propagation has been presented through textbooks, classroom discussion and laboratory experiments. These methods, when combined with intensive efforts on the part of the student to derive and apply acoustical concepts results in proven academic accomplishments.

However, in today's academic environment, students have access to high performance computing facilities which can greatly augment the learning process. The modeling and simulation capability of modern engineering software will never replace a well equipped laboratory, but it can help illustrate and analyze complex physical principles which may seem obscure on the printed page or on the chalkboard yet are challenging or inconvenient to duplicate in the laboratory.

B. PURPOSE

This thesis provides computer algorithms which examine selected topics drawn from the text, Fundamentals of Acoustics, Third Edition, John Wiley & Sons, Inc., by Kinsler, Frey, Coppens, and Sanders, (KFCS).

This is not a passive recitation of acoustic phenomena, but complements KFCS with interactive student participation. The figures included in this thesis are small, and it can be difficult to decipher important details. Their purpose is to support descriptions of what the student should see in the MATLAB figure window. The figure window will show detailed, color graphs and images which can even be zoomed in on with the software's zoom command. Seeing the results of manipulating variable ranges and observing animations is fundamental to successful implementation of these algorithms.

C. ESSENTIAL REQUIREMENTS

1. Text

The first paragraph of each chapter in this thesis refers the reader to specific sections of KFCS for the necessary background material and physics development. It is vital that the student have a fundamental knowledge of the topic under consideration before attempting to understand the output of the applicable algorithms. As indicated above, this thesis merely provides supporting material to the acoustics instruction provided by KFCS.

2. Software

The technical computing software, MATLAB™, Professional Version 4.2 or greater is required to run the algorithms. In most cases, the student edition of MATLAB will be acceptable, however, there are several differences between the student and professional editions.

The foremost difference is that the student edition cannot handle a vector of greater than 8192 elements or a matrix of greater than 32 rows or columns. In several of the accompanying algorithms, such limitations would cause poor graphics resolution and may effect numeric accuracy.

For additional information about MATLAB, call or write The MathWorks, Inc., University Sales Department, 24 Prime Part Way, Natick, Massachusetts 01760-1500, at (508) 653-1415. MathWorks can also be reached at its World Wide Web site at http://www.mathworks.com.

D. FILE NAME CONVENTION

1. Script Files

The vast majority of accompanying algorithms are script files or simple text files that tell MATLAB how and when to execute specific commands. These files are named according to the sections of KFCS which they support. For example, KFCS Chapter 2 Section 7 is supported by three script files, s02s07.m, c02s07a.m, and c02s07b.m. The m-extension at the end of each filename is a convention required by MATLAB. Script files are often called M-files.

The key to understanding the concepts discussed in this thesis is in the interactive nature of the accompanying algorithms. Each chapter suggests modifications to the algorithms which will aid in manipulating the figure or data output. These modifications can be accomplished with any standard text editor or using the Open M-file option in the File menu which is available in the MATLAB command window.

2. Function Files

A small number of files on the diskette are function files. These files are similar to M-files except that the variables they work with are not generally available to the user through the command window. In this thesis, function files usually perform services for several script files which if not used would cause the script files to be extremely large or unwieldy. Function files are, hopefully, named so that the user can intuitively assume their use. For example, the function file bsl.m determines Bessel zero crossings and extrema.

E. DISK CONTENTS

By providing a blank diskette, a copy of all algorithms can be obtained from Professor James V. Sanders at the following address: Physics Department (Code PH), Naval Postgraduate School, 833 Dyer Road, Room 203, Monterey, California 93943-5117.

The diskette is arranged in seven subdirectories. Each subdirectory contains algorithms supporting the appropriate chapter in KFCS. For example, subdirectory C01 contains algorithms supporting KFCS Chapter 1. A summary of subdirectory contents is provided below:

- C01: fundamentals of vibration
- C02: transverse motion
- C03: vibrations of bars
- C04: vibrations of membranes and plates
- C05: acoustic waves
- C06: transmission phenomena
- C08: radiation and reception of acoustic waves

F. TYPOGRAPHIC STYLES

Equations and equation variables are printed in italics to set them apart from the accompanying text.

Double quotes surround algorithm variables and lines of code that are addressed in the text. For example, when it is recommended that an algorithm be edited to allow a different range of values for a particular variable, such as "x", the suggested line of code may look like this: "x = 1:10".

II. SIMPLE HARMONIC OSCILLATOR

Algorithm c01s02.m produces a plot of the displacement and velocity of a simple harmonic oscillator. This program and the following discussion illustrate the theory and concepts addressed in KFCS Section 1.2.

A. CONCEPT

In Figure 2.1, a discrete sampling of the displacement of a simple harmonic oscillator is plotted as small circles and arrows indicate the direction and magnitude of the velocity at each position. An arrow pointing up indicates that the velocity is upward, and an arrow pointing down indicates the velocity is downward. In later plots we will explore conditions in which the velocity vectors point have horizontal components.

B. SALIENT FEATURES OF c01s02.m

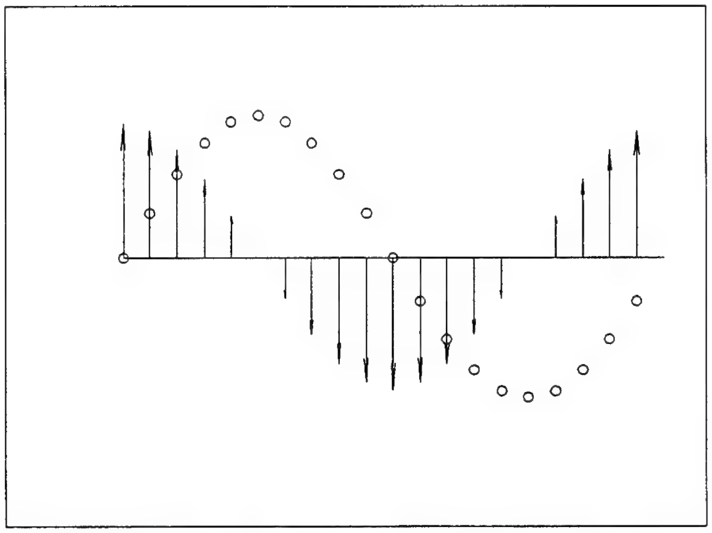

Figure 2.1 Displacement and Velocity of Simple Harmonic Oscillator

1. Amplitude

In a simple harmonic oscillator, the frequency of vibration is independent of the amplitude of vibration (KFCS Section 1.2). In this code, the variable "A" contains the amplitude of vibration. For example, "A=2", will effect the magnitude of displacement and velocity.

2. Frequency and Period

The frequency of vibration, "f", and the period, "T", are inversely proportional, T = 1/f. Note that the plot above contains one complete period of oscillation. To display more periods or waves, decrease the value of "T". For example, changing "T" to "T=max(t)/2" will cause two complete periods to be displayed.

C. ALGORITHM c01s02.m

```
% c01s02.m  simple harmonic oscillator
% plot of displacement and velocity of simple harmonic oscillator
clear, figure(1), clg
A=1;                        % amplitude of vibration
t=0:20;                     % time axis
T=max(t);                   % period
w=2*pi/T;                   % angular freq
x = A.*sin(w*t);            % vertical displacement
v = w*A*cos(w*t);           % velocity
plot(t+1,x,'o'), hold on
```

```
axis([min(t),T,-1.5,1.5]), axis('off')

feather(zeros(size(v)),v*3)
```

III. INITIAL CONDITIONS

Algorithm c01s03.m plots displacement of an undamped oscillator for various initial conditions. This program and the following discussion illustrate concepts addressed in KFCS Section 1.3.

A. CONCEPT

The solution of any ordinary differential equation is completely determined by its initial (or boundary) conditions. The equation of motion for an undamped, simple harmonic oscillator is expressed as a linear, second-order, ordinary differential equation, KFCS Equation 1.3. As such, the solution containing two arbitrary constants determined by the initial values of velocity and displacement.

Figure 3.1 was generated by algorithm c01s03.m and graphs the displacement of a

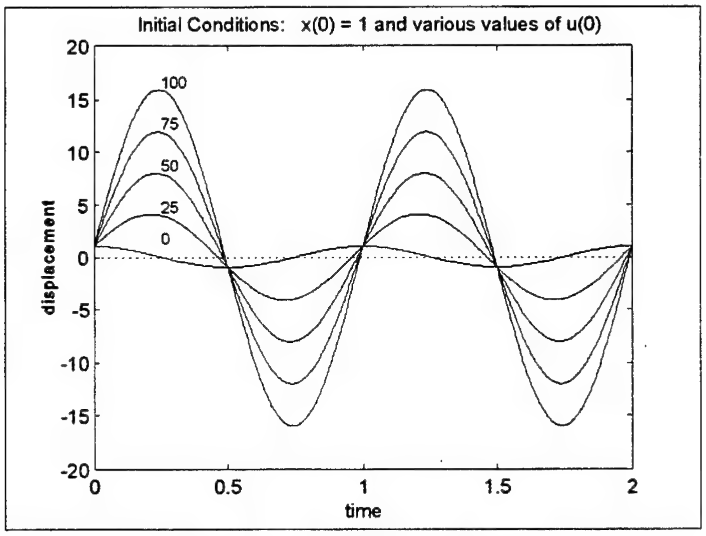

Figure 3.1 Initial Conditions

simple harmonic oscillator for an initial displacement of unity and various values of initial velocity.

B. SALIENT FEATURES OF c01s03.m.

Initial conditions describe the state of the system at time zero.

1. Initial Displacement

The initial displacement is contained in the constant “x0". For “x0 = 1", all of the displacement curves begin at x = 1. Changing the initial displacement to “x0=10", will cause a dramatic change in the appearance of the displacement curves, but their relationships will remain the same. All curves will begin at the same initial displacement, they will retain their same relative maximum amplitudes, they will all have the same periods of oscillation, and will all be at x = 1 or x = x0 after an integral number of oscillations.

2. Initial Velocity

The various initial velocities are contained in the vector “u0". Any number of initial velocities can be plotted. For example, changing the initial velocity vector to “u0 = 0:25:1000", will cause 41 displacement curves to be plotted for initial velocity values ranging from 0 to 1000. If over five curves are plotted or the initial displacement is greater than one, text with the values of initial velocity is not printed on the graph to avoid interference with displacement curve plots.

C. ALGORITHM c01s03.m

```
% c01s03.m  Initial conditions

% Plots displacement of an undamped oscillator for various initial conditions

clear, figure(1), clf

x0=1;                                  % initial displacement

u0=0:25:100;                           % initial velocity

w=2*pi;                                % angular frequency

t=linspace(0,2,1000);                  % time

[U0,T]=meshgrid(u0,t);

x=x0*cos(w*T)+U0/w.*sin(w.*T);         % KFCS Equation 1.5

plot(t,x)

if abs(x0)<=1 & length(u0)<=5

  for n=1:length(u0)

    text(w/8/pi,max(x(:,n))+.7,num2str(u0(n)),'fontsize',10)

  end

end

title(['Initial Conditions:   x(0) = ',num2str(x0),' and various values of u(0)'])

xlabel('time'), ylabel('displacement')

line([min(t),max(t)],[0,0],'linestyle',':','color',[1 1 1])
```

IV. COMPLEX EXPONENTIAL

Algorithm c01s05.m plots the motion of $e^{j\omega t}$ on the complex plane. This program and the following discussion illustrate the theory and concepts addressed in KFCS Section 1.5.

A. CONCEPT

Another solution to the equation of motion for a simple harmonic oscillator involves the use of the complex exponential. Values of $e^{j\omega t}$ are represented in the Figure 4.1 as phasors, i.e., arrows originating at the center of a circle of unit radius. The real axis is horizontal and the imaginary axis is vertical. As each phasor is plotted, the value of its real and imaginary projection is displayed and an "x" is left to mark its real component. Note that the two phasors that are mirror images in the x-axis have the same

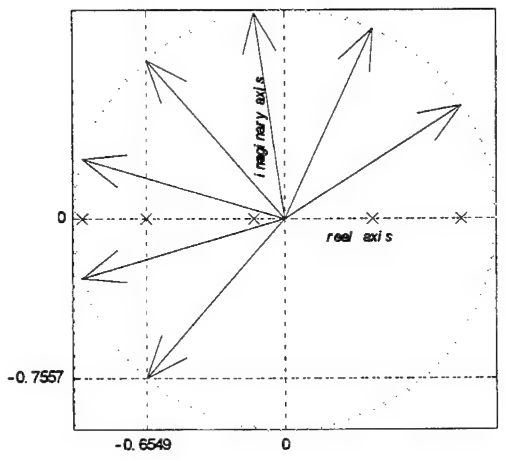

Figure 4.1 Complex Plane

B. ALGORITHM LIMITATIONS

It is not apparent in this plot when a phasor has rotated farther than one complete cycle, or period. For example, the plot of a phasor that has rotated through π radians will be identical to the plot of a phasor that has rotated through 9π radians.

C. SALIENT FEATURES OF c01s05.m

1. Phasor Position

This program is currently set up to plot eleven equally spaced phasors. Phasors of any position can be plotted by modifying the value of the "wt" vector. An example is provided in the program which plots vectors at 0, 30, 60, 90, and -45 degrees. Simply remove the "%" symbol to plot these values. Note that MATLAB considers angles only in radians, not degrees. That's why the vector is multiplied by the conversion factor pi/180.

2. Phasor Magnitude

Phasor magnitude is determined by the constant "A". Currently this constant is set to "A=1" so that a unit amplitude vector will be displayed. "A" can be set to any real value to observe the direct relation between amplitude and the value of the real and imaginary components. However, if you make the value of "A" greater than unity you will also need modify the size of axis displayed. For example, if "A" is changed to "A=2", the axis will need to be modified to "axis([-2 2 -2 2])" so that the entire vector may be seen when plotted. Should "A" be changed to a complex value, such as "A=1+0.5j" or "A=2*exp(pi/3)", the phase of the complex "A" will be added to the phase of the phasor and the position as well as the magnitude of the phasor may be changed.

D. ALGORITHM c01s05.m

```
% c01s05.m  Complex Exponential.  Plots exp(jwt) on the
% complex plane as well as its projection on the real axis
clear, figure(1), clg
w=2*pi/11;                          % angular frequency
t=1:11;                             % time
A=1;                                % amplitude
wt=w*t;
% wt=[0,30,60,90,-45].*pi/180;
c=A.*exp(j*wt);                     % complex exponential
plot(cos(linspace(0,2*pi,100)),...
sin(linspace(0,2*pi,100)),'.');     % unit circle
text(-.1,.2,'imaginary axis','fontangle','italic',...
        'fontsize',11,'rotation',90)
text(.2,-.1,'real axis','fontangle','italic','fontsize',11)
hold on
for n=1:length(c);
  compass(real(c(n)),imag(c(n)))
  axis('square'), axis([-1,1,-1,1])
  plot(real(c(n)),0,'xg')
```

```
 set(gca,'xtick',[0,real(c(n))],'ytick',[0,imag(c(n))])
 grid on
pause(2)
end
hold off
```

V. DAMPED HARMONIC OSCILLATOR

Algorithm c01s06.m plots displacement over time for a harmonic oscillator for various values of the temporal absorption coefficient, β. This program and the following discussion illustrate the theory and concepts addressed in KFCS Section 1.5.

A. CONCEPT

Damping of free oscillations describes a decrease in amplitude with time. Including damping in the equation of motion brings in terms modifying the first time derivative of displacement, velocity. The solution to this equation introduces the coefficient β, which describes the frequency in which the amplitude of oscillation is reduced to *1/e* of its initial value.

Each of the graphs in Figure 5.1 illustrate free oscillation at the same natural angular frequency, ω_o, but with different values of β. It's important to consider both β and ω_o when comparing the graphs, because the more meaningful description of decay is not just in β, but in the ratio of β to ω_o.

When β is less than ω_o, the natural angular frequency of the damped oscillator, ω_d, is a real valued constant. In this condition, the oscillator is said to be underdamped. After running algorithm c01s06.m, the zoom function of MATLAB will be turned on. Use the left mouse button to zoom in on the maximum amplitude of one of the first oscillations, and notice the relationship between the maximum displacement and the decaying

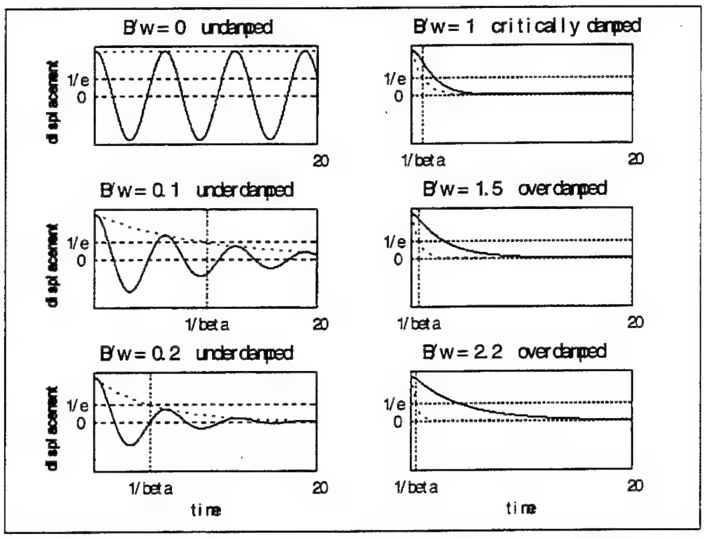

Figure 5.1 Displacement Over Time of a Damped

Harmonic Oscillator

decaying exponential.

When β is equal to ω_o, the system is critically damped. ω_d is equal to zero. Instead of oscillating, displacement decays exponentially.

When ω_o is less than β then ω_d is complex. In this condition, the system is overdamped, and displacement decays at a rate less than that of the decaying exponential.

B. SALIENT FEATURES OF c01s06.m

1. Temporal Absorption Coefficient

The values of β are contained in the vector "B". This vector can be modified to contain any number of positive, real values. The program will automatically determine the ratio of β to ω_o, and will display that ratio in the plot title. The program will also modify the number of subplots displayed to match the number of β values.

The position along the horizontal axis of 1/β is displayed show the intersection of 1/β with the 1/e amplitude.

2. Initial Conditions

The initial displacement of each plot is set to a value of unity and the initial velocity is set to zero. This was done to make the graphs easy to compare without having to normalize. The algorithm will accept any initial conditions and give a correct representation, but you may not be able to see the plot unless you turn off the axis constraints.

The value of ω_o is set to unity, but can be modified as desired.

C. ALGORITHM c01s06.m

```
% c01s06.m  Damped harmonic oscillator

% plots displacement over time

clear, figure(1), clg

beta=[0 1 .1 1.5 .2 2.2];             % temporal absorption coeff

w0=1;                                 % natural freq

x0=1;                                 % initial displacement

u0=0;                                 % initial velocity

t=linspace(0,20,1000);                        % time

t_exp=linspace(min(t),max(t),50);

for n=1:length(beta)

  wd=sqrt(w0^2 - beta(n)^2);          % damped freq
  if beta(n)<w0, condition='underdamped';
  if beta(n)==0, condition='undamped'; end
  phi=atan2(-((u0/x0)+beta(n)),wd);
  A=x0/cos(phi);
  x=A*exp(-beta(n)*t).*cos(wd*t+phi);
elseif beta(n)==w0, condition='critically damped';
  x=(x0+(u0+beta(n)*x0)*t).*exp(-beta(n)*t);
elseif beta(n)>w0, condition='overdamped';
  alpha1=beta(n)-abs(sqrt(beta(n)^2-w0^2));
  alpha2=beta(n)+abs(sqrt(beta(n)^2-w0^2));
  A1=(u0+alpha2*x0)/(alpha2-alpha1);
  A2=x0-A1;
  x=A1*exp(-alpha1*t)+A2*exp(-alpha2*t);
end
subplot(ceil(length(beta)/2),2,n)
plot(t,real(x),t_exp,exp(-beta(n).*t_exp),':w')
title(['B/w = ',num2str(beta(n)/w0),'  ',condition],'fontsize',10),
set(gca,'ytick',[0,1/exp(1)],'yticklabels',[' 0 ';'1/e'],...
 'xtick',[1/beta(n),20],'xticklabels',...
 ['1/beta';'  20  '],'fontsize',8)
axis([0 max(t) -1.1 1.1]), grid
```

```
 if n==5|n==6, xlabel('time'), end
 if n==1|n==3|n==5, ylabel('displacement'), end
end
zoom on
```

VI. FORCED OSCILLATIONS

Algorithm c01s07.m shows the response of a square wave driving a damped harmonic oscillator. This program and the following discussion illustrate the theory and concepts addressed in KFCS Section 1.7.

A. CONCEPT

This algorithm employs convolution theory to determine the response of an oscillator to a square wave. The upper graphs in Figure 6.1 plot the square wave driving function. The middle graphs plot the displacement of a damped harmonic oscillator to an impulse. Each of the bottom graphs plot the response of the oscillator to the driving square wave.

In convolution theory, a periodic input is assumed to have been applied since the

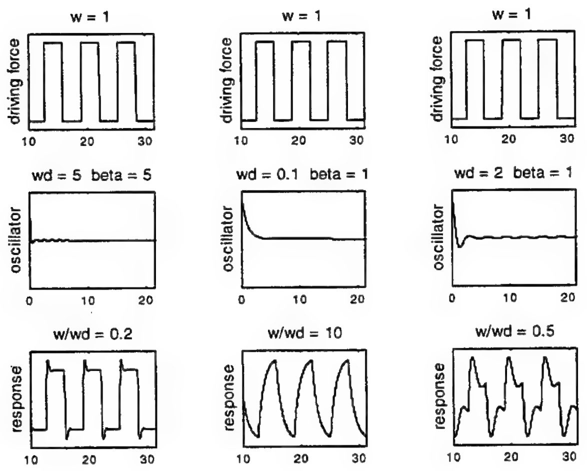

Figure 6.1 Square Wave Driving a Damped Harmonic Oscillator

infinite past and will remain on until the infinite future. Unfortunately, in this algorithm, as in real life, the forcing function is causal and finite; it begins at time zero and proceeds for about 10π seconds. It takes the system a little time to settle out and display the anticipated response. To avoid confusion, the first ten seconds of the response are not displayed. The transient response of an oscillator is discussed in more detail in the next chapter.

B. SALIENT FEATURES OF c01s07.m

1. Frequency Ratio

The ratio of the frequency of the driver, "w", to that of the damped oscillator, "wd", is an important factor in evaluating the response the system. The ratio w/wt is displayed above each response graph. To observe and compare the effects of changing either "wd" or "w", change the values of the vectors identified as "w" and "wd". These vectors currently contain three values each. It doesn't matter how many values the vectors contain as long as they both have an equal number. However, it's recommended that no greater than four values are used, otherwise the plots become too small to be useful. The program will automatically change the number of subplots to coincide with the length of these vectors.

2. Temporal Absorption Coefficient

The decay rate of the oscillator also plays an important factor in the system response. Vary the decay rate by changing the value of the temporal absorption coefficient, β. These values are contained in the vector, "B". Make sure that this vector contains the same number of values as the driver and transient (oscillator) frequency vectors described above.

3. Zoom

The last line in this algorithm turns on MATLAB's zoom function. This is very handy in evaluating the response graphs. The left mouse button causes the graph to zoom in at the position of the pointer and the right mouse button causes the graph to expand back out.

C. ALGORITHM c01s07.m

```
% c01s07.m  Forced Oscillations.
% Damped harmonic oscillator driven by a square wave
clear, figure(1), clg
w=[1 1 1];                  % driver freq
wd=[5 .1 2];                % damped oscillator freq
B= [5 1 1];                 % absorption coeff (beta)
t=linspace(0,10*pi,1000);
for n=1:length(w)
  driver=(2*rem((w(n)*t<0)+(rem(w(n)*t,2*pi)>pi |...
    rem(w(n)*t,2*pi)<-pi)+1,2)-1);     % square wave
  transient=exp(-B(n).*t).*cos(wd(n)*t);
  response=real(conv(driver,transient));
  response=response./max(response);
```

```
 subplot(3,length(w),n)
  plot(t,driver)
  axis([10 max(t) -1.2 1.2])
  ylabel('driving force','fontsize',10)
  title(['w = ',num2str(w(n))],'fontsize',10)
  set(gca,'ytick',[],'xtick',[0 10 20 30 40],'fontsize',8)
 subplot(3,length(w),n+length(w))
  plot(t,transient)
  axis([0 max(t)-10 -1.2 1.2])
  ylabel('oscillator','fontsize',10)
  title(['wd = ',num2str(wd(n)),'  beta = ',num2str(B(n))],'fontsize',10)
  set(gca,'ytick',[],'xtick',[0 10 20 30 40],'fontsize',8)
 subplot(3,length(w),n+length(w)*2)
  plot(t,response(1:length(t)))
  axis([10 max(t) -1.2 1.2])
  ylabel(['response'],'fontsize',10)
  title(['w/wd = ',num2str(w(n)/wd(n))],'fontsize',10)
  set(gca,'ytick',[],'xtick',[0 10 20 30 40],'fontsize',8)
end
zoom on
```

VII. TRANSIENT RESPONSE OF AN OSCILLATOR

Algorithm c01s08.m shows the transient response of a damped harmonic oscillator excited by a sine wave driving force. This program and the following discussion illustrate the theory and concepts addressed in KFCS Section 1.8.

A. CONCEPT

As in the previous chapter, this program employs convolution theory to superimpose a driving force on a damped oscillator. The techniques involved are identical to those used in c01s07.m. The difference being that the intermediate plots of the driving force and the oscillator are not displayed.

In this program, we examine the displacement of a resting oscillator immediately following the application of a driving force, turned on at time, t = 0. The effects of the homogeneous solution of the equation of motion, $Ae^{-\beta t}cos(\omega_d t+\varphi)$, KFCS Equation 1.33, are most apparent at this time, during the transient phase. As time progresses, the decaying exponential approaches unity and it's influence on oscillator motion becomes less important. When the transient effects become negligible, the system has achieved steady state: given by the particular solution $Fsin(\omega t-\varphi)/\omega Z_m$, also part of KFCS Equation 1.33.

Figure 7.1 shows the displacement of three oscillators with the same temporal absorption coefficient, β, but with a different ratio of driving frequency to damping oscillator frequency, ω/ω_d.

B. ALGORITHM LIMITATIONS

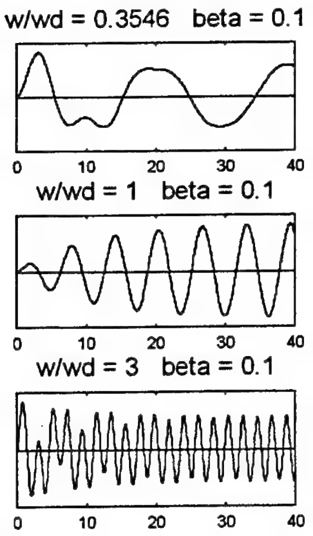

Figure 7.1 Transient Response of an Oscillator

The displacements by this algorithm are normalized so the amplitudes of the driver and oscillator are set to unity. Note, however, that varying these amplitudes will modify the height of the response curve but not change its shape.

C. SALIENT FEATURES OF c01s08.m

1. Frequency Ratio and Temporal Absorption Coefficient

The ratio of oscillator frequency, to driver frequency, and the value of the temporal absorption coefficient, play the same role in transient motion that they did in the steady state motion shown in the previous chapter. As in algorithm c01s07.m, this ratio can be adjusted by changing the values contained in the vectors “w”, “wd”, and “B”. These vectors can contain any number of values as long as they are all the same size. The program will automatically determine the correct number of subplots to display.

2. Time Display

The length of time currently displayed should be enough to get a good idea of the steady state motion of the oscillator for most frequency ratios and β. However, if a longer or shorter time axis is desired simply modify the time vector, "t". For example, to double the length of time displayed, change the time vector to "t=linspace(0,80,1000)". To remove the transient motion from the plot entirely, change the time vector to "t=linspace(20,50,1000)".

3. Impulse Response

The impulse response of the system gives a good insight into the time required for the oscillator to reach steady state. The impulse response, given by the homogeneous solution to the equation of motion, can be graphed by changing the plot command to "plot (t , transient)". This command is already included in the program and can be activated by removing the "%" symbol that precedes it.

D. ALGORITHM c01s08.m

```
% c01s08.m  Transient response of an oscillator.
% Damped harmonic oscillator driven by a sine wave
clear, figure(1), clg
w= [1/3 1 3];          % driver freq
wd=[.94 1 1];          % damped oscillator freq
B= [.1 .1 .1];         % absorption coeff (beta)
t=linspace(0,40,1000);
```

```
for n=1:length(wd)
 driver=sin(w(n)*t);
 transient=exp(-B(n).*t).*cos(wd(n)*t);
 response=real(conv(driver,transient));
 response=response./max(response);
 axes('Position',[.3 1-1/length(w)*n+.05 .4 1/length(w)-.12]);
 plot(t,response(1:length(t)))
 % plot(t,transient);   % impulse response
 axis([0 max(t) -1.2 1.2])
 title(['w/wd = ',num2str(w(n)/wd(n)),'   beta = ',num2str(B(n))])
 set(gca,'ytick',[0],'yticklabels',[],'ygrid','on',...
         'xtick',[0 10 20 30 40],'fontsize',8)
end
```

VIII. POWER RELATIONS AND MECHANICAL RESONANCE

Algorithm c01s09.m plots the average power and associated phase angles of forced oscillators with various quality factors and driving forces. This program and the following discussion are derived from the theory and concepts addressed in KFCS Sections 1.9 through 1.10.

A. CONCEPT

In this chapter, the term "power" will refer to time averaged power, as opposed to the "instantaneous power" supplied to a system at a particular instant. Figure 8.1 shows graphs of the relative power and phase angles of three different quality factors, Q, with the same driving force, F.

The quality factor is a dimensionless quality that describes the sharpness of a

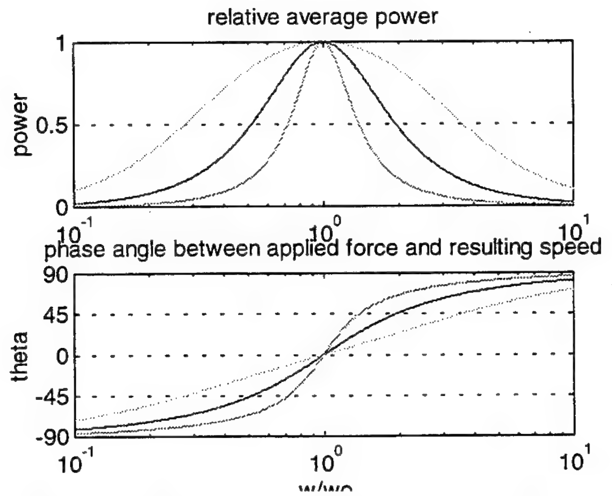

Figure 8.1 Relative Average Power for Various Quality Factors

resonance. In the plots of relative power, the oscillator with the highest Q has the sharpest resonance curve, while that with lowest Q has the flattest curve. In the phase angle plots, the highest Q is shown by the steepest slope at the resonance frequency, while the lowest Q has the flattest slope.

Quality factor can be determined by the ratio of resonance frequency to half-power bandwidth. Since the maximum power of each oscillator has been normalized to unity, the half power points in the graph are the frequencies at which the power curve intersects the dotted line at 0.5 on the power axis. The largest bandwidth is associated with the lowest Q oscillator. A similar method is used to determine quality factor in the phase angle plots. In this case, the half-power points are the intersection of the phase angle curves with ±45 degrees on the theta axis. Notice again that the largest bandwidth is associated with the lowest Q oscillator.

One method of calculating power is provided in KFCS equation 1.34, power = $F^2R(\cos\Theta)/(2Z)$, where F is the driving force, Z is the magnitude of the mechanical impedance, and theta is the phase angle between the driving force and the resulting speed. This equation is very useful in interpreting the power relations plotted in the figure. The power delivered to each oscillator is maximum when the driving frequency, ω, is equal to the resonant frequency of the oscillator, ω_o. The KFCS equation supports this observation since in also exhibits maximum power ω equals ω_o, since at this frequency, the phase is zero, so the cosine term is at its maximum, and the impedance is at its minimum value, since the reactive part has vanished.

B. SALIENT FEATURES OF c01s09.m

1. Quality Factor

The values of quality factor is modified by changing the values in vector, "Q". Any number of Q's can be entered and plotted. Since small Q's have a very broad power curve, the shape of the curve for very small Q's may not be adequately represented without also changing the length of the axis displayed.

2. Driving Force

In Figure 8.2, the relative average power was computed for a constant driving force but different quality factors. It is also useful to observe the effects of varying the driving force but maintaining the same Q. The figure below shows the result of changing the driving force through vector "F", but changing "Q" to a constant value. For example, "F=[.7 1 1.3]; Q=[1 1 1];".

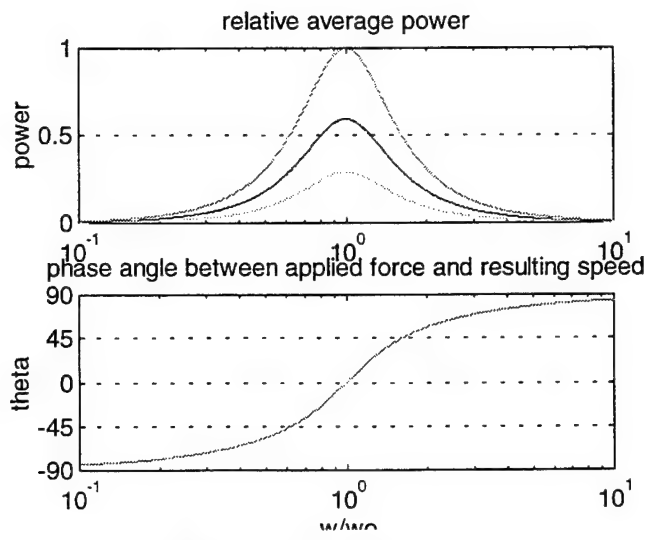

Figure 8.2 Relative Average Power for Various Driving Forces

C. ALGORITHM c01s09.m

```
% c01s09.m  Power relations and mechanical resonance

% Plots average power and associated phase angles

% for a forced oscillator with various Q values

clear, figure(1), clf

Q=[.3 .7 1.5];                        % quality factor

F=[1 1 1];                            % driving force

%Q=[1 1 1];

%F=[.7 1 1.3];

w=logspace(-1,1,1000);                % driving freq

[QQ,ww]=meshgrid(Q,w);

[FF,ww]=meshgrid(F,w);

Z=1+j*(ww-1./ww).*QQ;                 % impedance

theta=atan((ww-1./ww).*QQ);           % phase angle

pwr=FF.^2./2./abs(Z).*cos(theta);     % power

subplot(211)

  semilogx(w,pwr./max(max(pwr)))

  title('relative average power')

  axis([min(w) max(w) 0 1])

  set(gca,'ytick',[0 .5 1],'ygrid','on')

  ylabel('power')
```

```
subplot(212)
 semilogx(w,theta*180/pi)
 title(['phase angle between applied force ',...
         'and resulting speed'])
 set(gca,'ytick',[-90 -45 0 45 90],'ygrid','on')
 axis([min(w) max(w) -90 90])
 xlabel('w/wo'), ylabel('theta')
```

IX. EQUIVALENT CIRCUITS FOR OSCILLATORS

Algorithm c01s12.m generates graphs showing the mechanical impedance and velocity analogous to the electrical circuits of KFCS Figures 1.11 through 1.13. This program and the following discussion illustrate the concepts addressed in KFCS Section 1.11 and 1.12.

A. CONCEPT

Until this point, our concept of an oscillator has consisted of mass, stiffness and mechanical resistance. In some cases it is simpler and more effective to think in terms of inductance, capacitance and electrical resistance as they interact in an electrical circuit. This algorithm determines the equivalent mechanical impedance of the electrical circuits of KFCS Figures 1.11 through 1.13. It uses this impedance to plot the relationship of driving frequency to oscillator velocity and displacement. The point of this program is not to focus on methods of equivalent circuit determination, but to present an introduction to the results and design criteria that can be obtained by tailoring the way in which oscillator components interact.

For example, Figure 9.1 shows the velocity and displacement profiles of three different equivalent electrical circuits. The circuit components of capacitance, inductance and resistance are the same, but they have been arranged differently or deleted. Each circuit shows different results over the same frequency range.

The circuit in KFCS Figure 1.11 has a totally reactive impedance. As a result, the circuit serves as a notch filter in shutting off the oscillator's displacement at the resonant frequency. The circuit in KFCS Figure 1.12 shows a consistent, stable velocity over a wide range of frequencies. The circuit in KFCS Figure 1.13 shows a very strong response over a very narrow bandwidth. Each circuit has specific applications for which they particularly useful.

B. PROGRAM LIMITATIONS

The primary tool this program uses for determining velocity and displacement is the circuit impedance. The impedance calculations are valid only for the circuits presented in KFCS Figures 1.11 through 1.13. The response of other circuits may be plotted, but their impedance formulas must be determined and coded into the program.

The shape of the response curves presented are accurate for the conditions

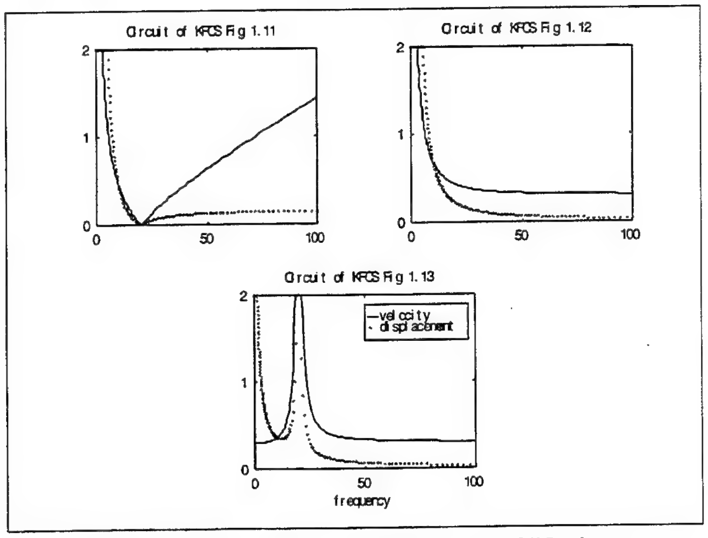

Figure 9.1 Frequency Response of Various Equivalent Electrical Circuits

described, but the actual magnitudes are scaled for easy circuit comparison.

C. SALIENT FEATURES OF c01s12.m

1. Stiffness-Controlled Regime

When the effects of stiffness dominate the frequency response of a system, it is said to be stiffness-controlled. The circuit in KFCS Figure 1.11 is a good example of this condition. Notice that at high frequency the displacement is independent of frequency, but the velocity is proportional. The value of stiffness can be changed by modifying variable, "s".

2. Resistance-Controlled Regime

The circuit in KFCS Figure 1.12 is a good example of a resistance-controlled system. Notice that in the upper frequencies of this circuit, the velocity is independent of frequency, but the displacement is inversely proportional. Change the resistance "R" and observe the difference.

3. Resonance Response

The circuit in KFCS Figure 1.13 shows the dramatic results that can be achieved by driving near the system resonance frequency. Resonance is a function of both mass and stiffness. The resonant frequency can be adjusted by changing the values of the mass, "m".

D. ALGORITHM c01s12.m

```
% c01s12.m  Equivalent Circuits

% velocity and displacement response of circuits in KFCS
% figures 1.11 through 1.13

clear, figure(1), clg

w=logspace(-1,2,250);         % driving frequency

m=.5;                         % mass

s=200;                        % stiffness

R=10;                         % resistance

F=3;                          % force applied

Zm=j*m*w;                     % impedance due to mass

Zs=-j*s./w;                   % impedance due to stiffness

%%%%%%% KFCS Figure 1.11

Z1=1./((1./Zm)+(1./Zs));      % impedance

U=F./abs(Z1);                 % velocity

A=U./w;                       % displacement

subplot(221), plot(w,U,w,A*10,'.')

axis([0,max(w),0,2])

set(gca,'ytick',[0 1 2],'fontsize',10)

title('Circuit of KFCS Fig 1.11','fontsize',10)

%%%%%%% KFCS Figure 1.12

Z2=1./((1./Zm) + 1./((1./Zs)+R));

U=F./abs(Z2);

A=U./w;
```

```
subplot(222), plot(w,U,w,A*10,'.')

axis([0,max(w),0,2])

set(gca,'ytick',[0 1 2],'fontsize',10)

title('Circuit of KFCS Fig 1.12','fontsize',10)

%%%%%%% KFCS Figure 1.13

Z3=1./((1/R)+1./(Zs+Zm));

U=F./abs(Z3);

A=U./w;

subplot(2,2,3.5), plot(w,U,w,A*10,'.')

axis([0,max(w),0,2])

set(gca,'ytick',[0 1 2],'fontsize',10)

title('Circuit of KFCS Fig 1.13','fontsize',10)

xlabel('frequency')

legend('velocity','displacement')
```

X. LINEAR COMBINATION OF SIMPLE HARMONIC VIBRATIONS

Algorithm c01s13.m plots the results of coherently combining two harmonic vibrations. Algorithm c01s13a.m produces the audible result of linear combination through the computer's sound system and plots the combined waveform and its envelope in the figure window. This program and the following discussion illustrate the concepts discussed in KFCS Section 1.13.

A. CONCEPT

Figure 10.1, produced by c01s13.m, shows the results of combining two vibrations of the same frequency, but different phases. One harmonic vibration is plotted with a dotted line, the other with a dashed line, and the linear summation of the two is drawn with a solid line. The title of each plot shows the ratio of the two frequencies,

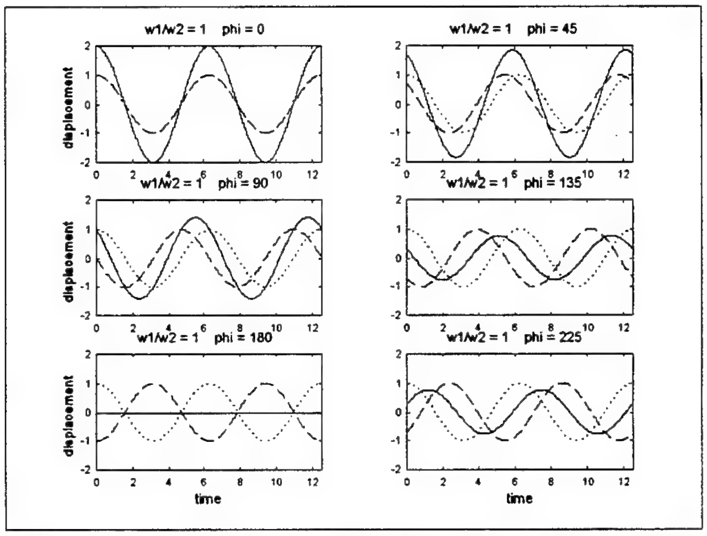

Figure 10.1 Linear Combination of Two Harmonic Vibrations

which is unity in each of these cases, and the phase difference in degrees.

When the vibrations are in phase, as in the upper left plot, the combined vibration shows the largest possible amplitude. The lower left plot shows the results of combining two waves 180 degrees apart. These vibrations cancel each other out.

Algorithm c01s13a.m coherently combines two harmonic vibrations at frequencies, f_1 and f_2, that are very close together. In this case, both frequencies begin at 262 hertz (middle C), but f_2 is incrementally increased until it's twice f_1. At each increment of f_2, the two vibrations are combined, and this combination is played over the computer's sound system.

As each new sound is produced, the figure window displays two plots, such as those in Figure 10.2, which will help to examine the sounds produced. The upper plot shows the result of the linear combination (the yellow line in the figure window) and the

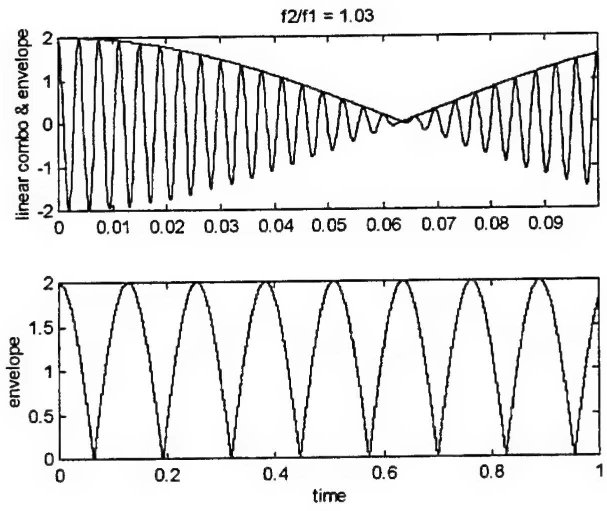

Figure 10.2 Linear Combination and Amplitude Envelope

amplitude envelope (purple line) over a period of 0.1 seconds. The lower plot shows only the envelope, but over a time period of one second. Two separate time indices were used because when the f_1 and f_2 are close together, it's difficult to observe the features of the envelope in the upper plot. When they are farther separated, the upper plot is very helpful, but the lower plot shows too many cycles to be useful.

KFCS Section 11.7(e) discusses the effects that this linear combination has on the sounds produced. When the two frequencies are very close together, you will hear a single frequency, $f_c = (f_1 + f_2)/2$, and will notice a very distinctive fluctuation in intensity. This fluctuation is termed beating, and occurs at the beat frequency, $f_b = |f_1 + f_2|$. You may also notice that the frequency of this beating is the same as the frequency of the envelope shown in the one second time interval of the lower graph.

When the ratio of f_1 and f_2 is the ratio of two integers, you will not detect beating. If they vary, even slightly, from integer numbers, beating will occur. KFCS Section 11.7(f), discusses fascinating tonal combinations that can be explored with algorithm c01s13a.m.

B. PROGRAM LIMITATIONS

There are interesting effects that can be studied by presenting separate tones to each ear (right and left speakers). Unfortunately, this feature is not currently available in MATLAB.

C. SALIENT FEATURES OF c01s13a.m

1. Period of Combined Vibration

Notice in Figure 10.2 that the period of the combined vibration is the same as that of the component vibrations, but phase shifted. This can be better observed by typing "plot(t,y)", in the MATLAB command window. All six combined vibrations will be graphed in the same plot. The variable, "y" contains a matrix with six columns, one for each of the six subplots.

2. More Than Two Component Vibrations

The principles of linear combination are not limited to just two components. As will be demonstrated in a following chapter, this is the basis of Fourier synthesis. To use this algorithm to add more than two waves, reduce the vectors "w1" and "phi1" to single elements, generate a new component vibration, such as "y3=cos(3*t)", and change "y=real(y1+y2)" to "y=real(y1+y2+y3)". These modifications will allow you to combine a small number of waveforms, but can quickly become tedious. Algorithm c01s14.m is a good example of the effects of combining large numbers of component vibrations.

D. SALIENT FEATURES OF c01s13a.m

1. Sound Duration

To change the length of time the sound of the incremental combination is produced, modify the variable, "T". For example, to play each sound for 3 seconds, use "T = 3".

2. Base Frequency and Incrementation

The base frequency, in hertz, is contained in the variable "f1" and may be changed to any value. The second frequency, "f2" is incremented by the addition of "dw". To increment "f2" by intervals for one hertz, for example, change the line of code that contains "dw" as follows: "for dw=0:1:10"

E. ALGORITHM c01s13.m

```
% c01s13.m Linear combinations

% Coherent sum of 2 sinusoidal waves

clear, figure(1), clf

w1=[1 1 1 1 1 1];                          % freq 1

phi1=[0 45 90 135 180 225]*pi/180;         % phase 1

t=linspace(0,4*pi,1000);                   % time

[Phi1,T]=meshgrid(phi1,t);

[W1,T]=meshgrid(w1,t);

W2=ones(size(W1));

Phi2=zeros(size(Phi1));

dw=W1-W2;                                  % freq difference

Phi=atan(sin(Phi1)+sin(Phi2+dw.*T)./(cos(Phi1)+cos(Phi2+dw.*T)+eps));

y1=exp(j*(W1.*T+Phi1));                    % wave 1

y2=exp(j*(W2.*T+Phi2));                    % wave 2

y=real(y1+y2);                             % combined waves
```

```
y1=real(y1); y2=real(y2);
for n=1:length(phi1)
  subplot(3,2,n)
  plot(t,y(:,n),t,y1(:,n),'- -',t,y2(:,n),':')
  axlim=max([max(y),max(y1),max(y2)]);
  axis([min(t),max(t),-axlim,axlim])
  title(['w1/w2 = ',num2str(W1(1,n)/W2(1,n)),...
    '    phi = ',num2str(phi1(n)*180/pi)],'fontsize',10)
  if n==1|n==3|n==5, ylabel('displacement','fontsize',10), end
  if n==5|n==6, xlabel('time','fontsize',10), end
  set(gca, 'fontsize',8)
end
```

F. ALGORITHM c01s13a.m

```
% c01s13a.m audible beating experiment
% with MATLAB's sound function
clear, figure(1), clf
T=2;                                    % sound length (seconds)
f1=262;                                 % base freq (hertz)
fs=8192;
N=T*fs;
t=linspace(0,T,N);
```

```
for dw=[0:f1/100:f1/10, f1*.2:f1/5:f1]
 f2=f1+dw;
 A=cos(2*pi*f1*t)+cos(2*pi*f2*t);          % linear summation
 B=sqrt(2+2*cos(2*pi*dw*t));               % beat amplitude
 ex1=[1:fs/10];                            % extent to plot 1
 ex2=[1:fs];                               % extent to plot 2
 subplot(211)
 plot(t(ex1),A(ex1),t(ex1),B(ex1))
 axis([t(1),t(max(ex1)),-2,2])
 title(['f2/f1 = ',num2str(f2/f1)])
 ylabel('linear combo & envelope','fontsize',12);
 subplot(212)
 plot(t(ex2),B(ex2),'color',[1 0 1])
 xlabel('time','fontsize',12)
 ylabel('envelope','fontsize',12);
 pause(.1)
 sound(A,fs)
end
```

XI. LINEAR COMBINATION OF INCOHERENT VIBRATIONS

Algorithm c01s13b.m plots the results of combining incoherent vibrations. This program and the following discussion employ concepts of incoherent combination discussed in KFCS Section 5.9, applied to displacement and power.

A. CONCEPT

When adding complex displacements, the result depends critically upon the phase difference between the component vibrations. Figure 11.1 shows the coherent sum of two in-phase vibrations. Within each plot, the power and the root mean square, rms, of displacement are given. The variables "d1", "d2", and "dc" refer to the first component, second component and combined displacements. The variables "p1", "p2", and "pc" concern power. The "% difference" is the between the arithmetic sum of the component

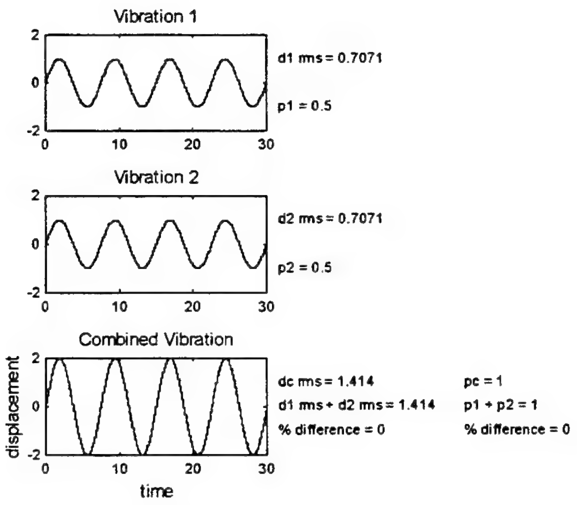

Figure 11.1 Coherent Combination

displacement or power and the actual value of the combined vibration.

The rms method expresses the mean of vibrations in a way that is more meaningful than the standard mean. For example, since a complete sinusoid contains an equal amount of positive and negative values, its mean is zero. However, its rms value is /·' or approximately 0.7071.

Note that for coherent vibrations, the percent difference is zero for both displacement and power.

In the Figure 11.2, two incoherent vibrations are combined. These vibrations are generated by adding a randomly oscillating phase to each time increment of a sinusoid. When these vibrations are combined, the rms displacement of the combined vibrations is not equal to the arithmetic sum of the component rms displacements. They differ by more that 25 percent. However, the combination of rms power of the combined vibrations is

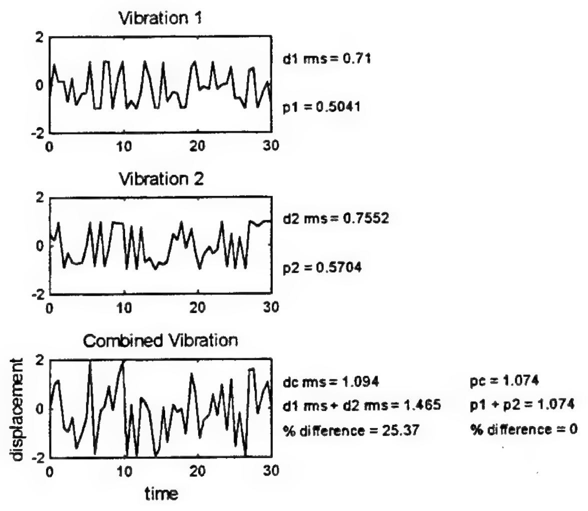

Figure 11.2 Incoherent Combination

the same as arithmetic component sum. This is because power is a time averaged value, the square of the rms displacement, while the displacement is an instantaneous value.

B. PROGRAM LIMITATIONS

The time averaged feature of acoustic power serves to smooth out irregularities, which makes it ideal for use with incoherent vibrations. To take advantage of this feature, we must have many time samples over which to average. To a degree, the same is true of the rms displacement values employed by this algorithm. This program uses discrete points to mimic the analog displacement vibrations found in nature. The more data points that are used, the more accurate the approximation to a continuous function. For example, consider the coherent component vibrations in the first figure. It took the averaging of a hundred thousand points to achieve the predicted rms displacement of 0.7071. If you were to perform the same calculations with only a hundred points, the rms displacement would be only 0.7036.

When creating the data vectors for the incoherent vibrations, a random phase was added to each time increment. To make the displacement plots more visually informative, only a hundred time increments were used. Therefore, the rms values vary from those that would have been obtained over more time increments.

C. SALIENT FEATURES OF c01s13b.m

1. Time Increments

Increasing the number of time increments to more effectively smooth out the rms displacement and power can be accomplished by changing the number of increments in vector “t”. For example, “t=linspace(0,60,1e5)”, will plot the number of points used in the first figure, 100,000.

2. Phase

The coherent vibrations in the Figure 11.1 have zero phase difference. To change the phase to zero, remove the percent signs before “phi1=0" and “phi2=0".

The incoherent vibrations employ a random phase between zero and 2*pi radians. To see the effects of giving the phase a smaller range of fluctuations, change the amplitudes of the “phi1" and “phi2" vectors. For example, “phi1=rand(size(t))”, changes the range of the phase to between zero and one radian. The effects of smaller phase is to cause the incoherent rms displacement to approach coherent values.

D. ALGORITHM c01s13b.m

```
% c01s13b.m  Linear combinations
% Coherent or incoherent sum of two sinusoidal vibrations
clear, figure(1), clf
t=linspace(0,30,50);
phi1=2*pi*rand(size(t));            % incoherent phase 1
phi2=2*pi*rand(size(t));            % incoherent phase 2
%phi1=0;                            % coherent phase 1
%phi2=0;                            % coherent phase 2
d1=sin(8*pi/max(t)*t+phi1);         % displacement 1
d2=sin(8*pi/max(t)*t+phi2);         % displacement 2
```

```
dc=d1+d2;                          % combined displacement

d1rms=sqrt(mean(d1.^2));

d2rms=sqrt(mean(d2.^2));

dcrms= sqrt(mean(dc.^2));

p1=d1rms.^2;                       % power 1

p2=d2rms.^2;                       % power 2

pc=p1+p2;                          % combined power

ddiff=num2str(abs((d1rms+d2rms-dcrms)/(d1rms+d2rms))*100);

pdiff=num2str(abs((p1+p2-pc)/(p1+p2))*100);

xp=max(t)*1.05;                              % x-position of text

subplot(321), plot(t,d1), title('Vibration 1')

 text(xp, 1,['d1 rms = ',num2str(d1rms)],'fontsize',10)

 text(xp,-1,['p1 = ',num2str(p1)],'fontsize',10)

subplot(323), plot(t,d2), title('Vibration 2')

 text(xp, 1,['d2 rms = ',num2str(d2rms)],'fontsize',10)

 text(xp,-1,['p2 = ',num2str(p2)],'fontsize',10)

subplot(325), plot(t,dc), title('Combined Vibration')

 text(xp, 1,['dc rms = ', num2str(dcrms)],'fontsize',10)

 text(xp, 0,['d1 rms + d2 rms = ',num2str(d1rms+d2rms)],'fontsize',10)

 text(xp,-1,['% difference = ',ddiff],'fontsize',10)

 text(xp*1.8, 1,['pc = ',num2str(pc)],'fontsize',10)
text(xp*1.8, 0,['p1 + p2 = ',num2str(p1+p2)],'fontsize',10)

text(xp*1.8,-1,['% difference = ',pdiff],'fontsize',10)

xlabel('time'), ylabel('displacement')

set([subplot(321),subplot(323),subplot(325)],'xlim',[min(t) max(t)],...

'ylim',[-2,2],'xtick',0:10:max(t)','fontsize',10)
```

XII. FOURIER SYNTHESIS

Algorithm c01s14.m displays the Fourier synthesis of a sawtooth or square wave. This program and the following discussion illustrate the concepts addressed in KFCS Section 1.14.

A. CONCEPT

As illustrated in the previous chapter on linear combination of coherent waves, the summation of harmonic vibrations can produce a combined wave of far greater complexity than its components. Fourier's theorem applies this principle in the synthesis of complex waveforms.

Figures 12.1 and 12.2 illustrate the synthesis of a square and sawtooth waveform by application of Fourier's theorem. "N" is the number of harmonic vibrations combined

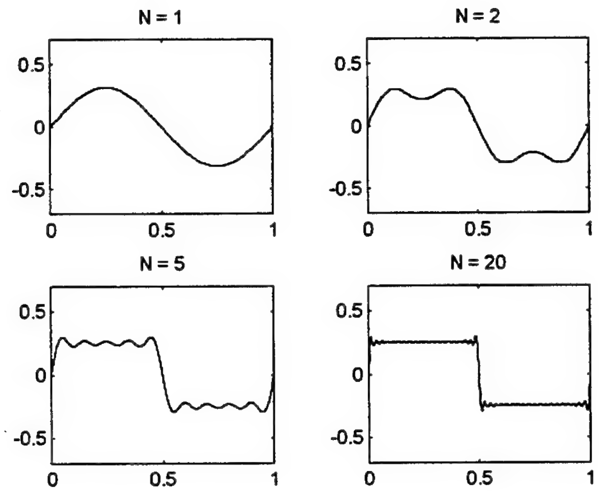

Figure 12.1 Fourier Synthesis of a Square Vibration

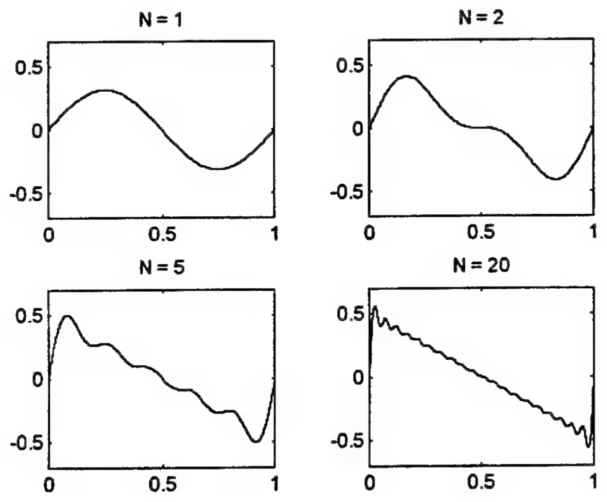

Figure 12.2 Fourier Synthesis of a Sawtooth Vibration

to create the waveform shown. The sawtooth vibration is depicted in KFCS Figure 1.15 and the square wave is analyzed in KFCS Problem 1.22.

B. SALIENT FEATURES OF c01s14.m

1. Number of Summations

Notice that as the number of component waves increases, the summation more closely resembles the desired waveform. With a finite number of terms we will never achieve an exact waveform match. Corners will be rounded or overshot and straight lines will not be straight.

The “zoom” command is turned on in this program. This will allow you to zoom in (left mouse button) and zoom out (right mouse button) to more closely examine different aspects of a plot. For example, zooming in on the corner of the N=20 square wave will give a good display of "Gaussian overshoot."

2. Discontinuities

An advantage to simulating continuous waveforms by discrete methods is that you can get a insightful look at the inability of Fourier synthesis to represent discontinuities. An ideal square wave, for example, jumps from its positive maximum amplitude to its negative maximum amplitude passing through intermediate values instantaneously. Since a Fourier synthesis is composed of continuous waves, it cannot reproduce a discontinuity.

Waveform near discontinuities can be readily observed by increasing "N" to an extreme value such as 20,000 and plotting the result as discrete points rather than a continuous line. To plot in discrete points, change the plot command to "plot(t,f,'.')". After plotting in this manner, notice how few, if any, points lie at the discontinuity position, while the positive and negative maximum amplitudes are densely packed.

3. Waveform

An interesting consequence of synthesizing sawtooth and square waveforms in this program is that they both use the same equation, identified as vector "f". The difference in generating the two waveforms is the way "n" is incremented. The Fourier series for a sawtooth waveform sums all positive integer values of "n", but for a square waveform the sum is over odd positive integers. Other waveforms sum even integers. To generate all integers use the command "for n=1:N(x)". For odd integers use "for n=1:2:2*N(x)-1". To sum even integers us "for n=2:2:2*N".

Changing to a waveform other than the two discussed above only requires modification of the equation of "f" and determining whether even, odd or all integers are summed.

4. Time Interval

The two featured waveforms both were generated for a unit period. To display more than the one period, change the maximum value of the time vector, “t”. For example, “t=linspace(0,3,1000)” will generated three complete periods.

C. ALGORITHM c01s14.m

```
% c01s14.m  Fourier Synthesis of a square and sawtooth waveform

clear, figure(1), clf

N=[1 2 5 20];                 % number of summations

w=2*pi;                       % angular freq

t=linspace(0,1,1000);         % time

for x=1:length(N)

 f=0;

 %for n=1:N(x)                 % all integers (sawtooth)

 for n=1:2:2*N(x)-1            % odd integers (square)

  f=f+sin(n*w*t)/n/pi;         % waveform vector

 end

 subplot(2,2,x)

 plot(t,f)

 title(['N = ',num2str(N(x))])

 axis([min(t) max(t) -.7 .7])
```

```
end

zoom on
```

XIII. FOURIER ANALYSIS

Algorithm c01s15.m plots the Fourier spectrum of a square pulse. This program and the following discussion illustrate concepts addressed in KFCS section 1.15.

A. CONCEPT

Figure 13.1 contains a graph of a square pulse in the time domain and its spectrum, a sinc function, in the frequency domain. The spectrum shows the results of resolving the pulse into its individual frequency components. It was obtained by forming the discrete Fourier transform of the pulse. The Fourier transform and its inverse allow you to go back and forth between the time domain and frequency domain.

The relative amplitudes of the individual frequencies making up the spectrum are the amplitude weights applied to the harmonic components which combine to form the

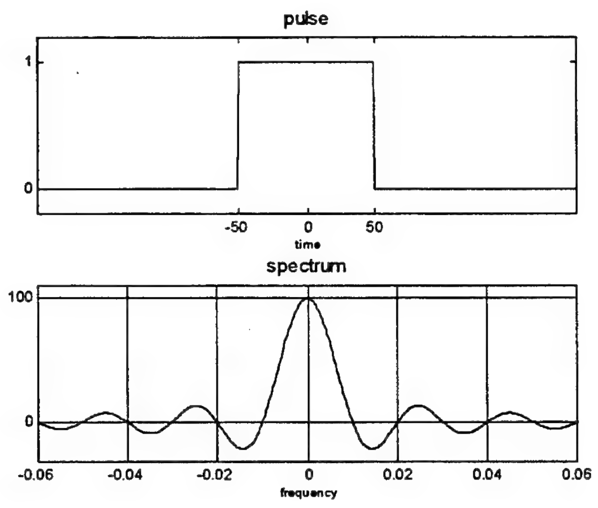

Figure 13.1 Square Pulse and its Spectrum

square pulse.

B. ALGORITHM LIMITATIONS

This algorithm does not focus on issues of sampling theorem, rather on aspects of Fourier analysis. Consequently, the assumption is made of a one time unit sampling period, such as one second or one millisecond. The sampling frequency, which is the inverse of the sampling period, is therefore one inverse-unit, such as one hertz or one kilohertz. Sampling theory will play an important role in applications which must resolve frequencies that are very close to one another, such as when examining Doppler shifts.

C. SALIENT FEATURES OF c01s15.m

1. Transform Pairs

The Fourier transform of a square pulse will always yield a sinc function. Similarly, the inverse Fourier transform of a sinc will always produce a square pulse. The sinc and the square pulse are, therefore, Fourier transform pairs. Many such pairs are well known and documented. Modifying the algorithm to consider other forms, such as a triangular pulse, will only require modification of the "pulse" vector. For example, changing the vector to "pulse=triang(Tp)';" will use MATLAB's triangular window function to generate a triangular pulse. Note the use of a single quote to change the triang function to a horizontal vector.

2. Relationship Between Pulse Size and Spectrum Size

It's interesting to note the relationship between the duration of the pulse and the bandwidth of the sinc function. The width of the pulse, in this case 100 units, determines the maximum height of the sinc, 100 inverse-units. The inverse of the pulse width, 1/100, determines the position of the first node (frequency of zero amplitude) of the sinc, 2/100 is the second node, 3/100 is the third, etc.. To modify the pulse width, change the variable "Tp". For example, "Tp=300" will generate a pulse of 300 time units in length.

Doubling the amplitude of the pulse will double the maximum height of the sinc. To modify the amplitude of the pulse, change the variable "A". For example, "A=2" doubles the amplitude of the pulse.

3. Relationship Between Pulse Shape and Spectrum Shape

While the relationship between pulse width and spectrum height is interesting, the primary concern usually lies in the relationship between the shape of the pulse and the relative heights of the main lobe of the spectrum (the region between the first positive and negative nodes) and its side lobes (the regions between higher order nodes).

Change the square pulse to a triangular one as indicated above and note the change in pulse width and sidelobe height. These proportions are vitally important in many design applications, but will not be discussed at this point.

4. Spectrum Phase

Normally, reference is made to a spectrum's absolute magnitude, but in this figure only the real part of the spectrum is plotted. An absolute magnitude plot would not adequately display the characteristic sinc shape. To plot the absolute magnitude of the spectrum, change the "fft_pulse" vector from "fft_pulse=real(...)" to "fft_pulse=abs(...)".

Notice that the pulse is centered about the zero of the time axis. Fourier theory dictates that shifting the pulse in the time domain will require a phase shift in the frequency domain equal to exp($-j\omega t_o$), where t_o is the amount time and direction of pulse shift. Except in unique situations, the Fourier transform always includes complex elements. The phase of the spectrum contains important information required for reconstructing the time domain pulse, though it is often disregarded.

D. ALGORITHM c01s15.m

```
% c01s15.m  Fourier Analysis: spectrum of a square pulse
clear, figure(1), clf
Tp=100;                  % pulse period (length)
A=1;                     % pulse amplitude
pulse=ones(1,Tp);        % square pulse
%pulse=triang(Tp)';      % triangular pulse
fft_pulse=real(fftshift(fft(pulse,Tp*100)));
subplot(211)
plot(-2*Tp:2*Tp,[zeros(1,3*Tp/2),pulse,zeros(1,3*Tp/2+1)])
title('pulse'), xlabel('time','fontsize',8)
axis([-2*Tp,2*Tp,min([0;pulse'])-.2,max(pulse)+.2])
set(gca,'fontsize',9,'ytick',[0,max(pulse)],'xtick',[-Tp/2,0,Tp/2])
subplot(212)
f=[-length(fft_pulse)/2:length(fft_pulse)/2-1]/Tp/100*2;
```

```
plot(f,fft_pulse), grid

title('spectrum'), xlabel('frequency','fontsize',8)

axis([-6/Tp,6/Tp,min(fft_pulse)-Tp/10,max(fft_pulse)+Tp/10])

set(gca,'fontsize',9,'xtick',[-6:2:6]./Tp,'ytick',[0,max(fft_pulse)])
```

XIV. PROPAGATION OF A TRANSVERSE DISTURBANCE ALONG A STRETCHED STRING

Algorithm c02s05.m animates the propagation of a transverse disturbance along a stretched string. This program and the following discussion illustrate concepts addressed in KFCS Sections 2.1 through 2.5.

A. CONCEPT

Figure 14.1 shows one frame of an animation. It portrays a string under tension which is suddenly displaced from equilibrium and released. It breaks up into two separate disturbances which propagate along the string in opposite directions. These right and left traveling waves are identical to the initial disturbance, but are half the amplitude. They propagate at equal speeds regardless of the shape or amplitude of the initial disturbance.

B. ALGORITHM LIMITATIONS

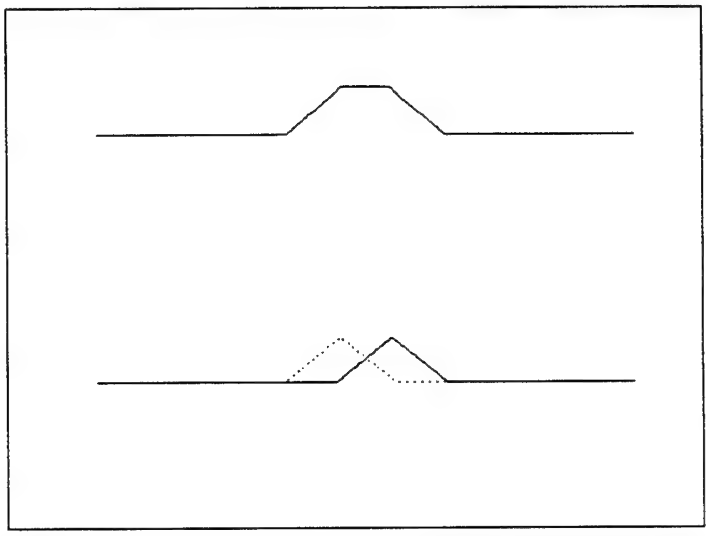
Figure 14.1 Propagation of a Transverse Disturbance Along a Stretched String

This animation stops before the right and left traveling waves reach the ends of the string. Boundary conditions will be addressed in the next chapter.

C. SALIENT FEATURES OF c02s05.m

Algorithm c02s05.m contains general code that will be modified for use in the following chapters. Features such as boundary conditions will be addressed later.

1. Wave Shape

The shape of the initial displacement is completely arbitrary. It can be modified by changing the values in the "shape" vector. For example, to describe the arc of a sinusoid, "shape=sin(linspace(0,pi,50));". This modification is included in the code, but not activated because of the preceding "%" symbol.

The shape and amplitude of the right and left traveling waves will be computed by the algorithm. Regardless of the shape of initial displacement, the components will be identical to the initial but half the amplitude.

D. ALGORITHM c02s05.m

```
% c02s05.m
% Animation of the propagation of a transverse disturbance
% along a stretched string
clear, figure(1), clf, set(gcf,'backingstore','off')
%%%%%%%% determining wave shape
shape=[1:25,25:-1:1];                  % triangle shape
%shape=sin(linspace(0,pi,50));         % sinusoidal shape
```

```
leftBC=+1;  rightBC=+1;

shape2=[zeros(1,100),shape,zeros(1,100)];

left=[rightBC*shape2,rightBC*leftBC*shape2,leftBC*shape2,shape2];

right=[shape2,rightBC*shape2,rightBC*leftBC*shape2,leftBC*shape2];

len=length(right);

%%%%%%% initializing plots

both=left(751:1000)+right(1:250);

subplot(211)

  Both=line(1:250,both,'erasemode','xor');

  axis([1 250 -max(2*shape2) max(2*shape2)]), axis('off')

subplot(212)

  Left=line(1:250,left(751:1000),'erasemode','background');

  Right=line(1:250,right(1:250),'linestyle',':',...

   'erasemode','background');

  axis([1 250 -max(2*shape2) max(2*shape2)]), axis('off')

%%%%%%% Plotting wave and moving components

for n=1:90

  both=left(751:1000)+right(1:250);

  set(Left,'ydata',left(751:1000))

  set(Right,'ydata',right(1:250))

  set(Both,'ydata',both)
```

```
 left=[left(len),left(1:len-1)];

 right=[right(2:len),right(1)];

 drawnow

end
```

XV. REFLECTION AT A BOUNDARY

Algorithm c02s07.m produces an animated plot of the interaction of a triangular wave and its reflection approaching a rigid or free boundary. Algorithm c02s07a.m uses similar programing to produce an animated plot of transverse waves traveling on a string with rigid or free boundary conditions at the string ends. The last algorithm, c02s07b.m uses the Fourier series solution of the transverse wave equation with free end conditions to the produce the same animation determined by algorithm c0207a.m for the free-free case. c02s07b.m is a simple, but inflexible algorithm developed to confirm the approach used in the first two. These programs and the following discussion illustrate concepts addressed in KFCS sections 2.6 and 2.7.

A. CONCEPT

Reflection at either a rigid or free boundary can be modeled as the summation of two identical traveling waves approaching one another at the boundary interface. When the boundary is fixed, the two waves are odd functions of one another.

Figures 15.1 and 15.2 are produced by c02s07.m. The dashed vertical line represents a boundary, either fixed or free. The solid line represents a wave traveling to the left, approaching the boundary. The dotted line represents a wave traveling to the right, approaching the boundary. Note that in reflection from a fixed boundary, the reflected wave is of similar shape but opposite displacement. The wave reflected from a free boundary maintains the same displacement.

If you use your hand or a sheet of paper to cover the left side of the display, you

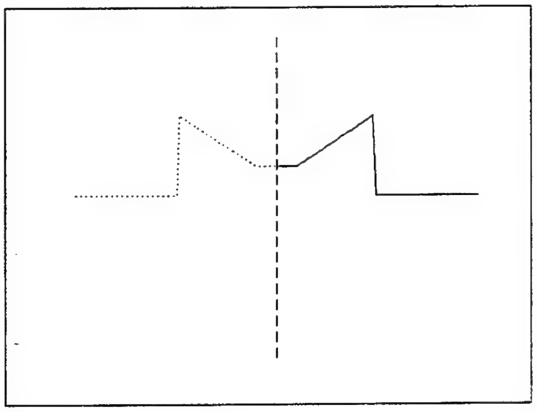
Figure 15.1 Free Boundary Interaction

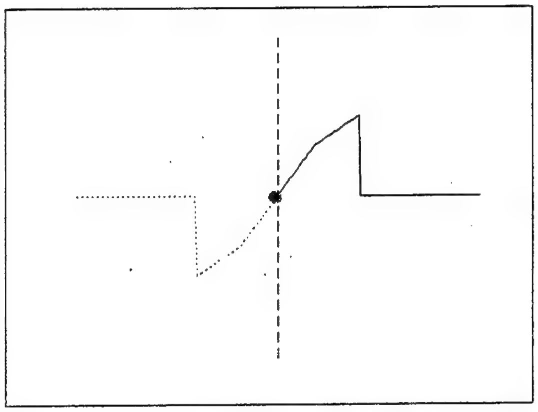
Figure 15.2 Fixed Boundary Interaction

can observe the process of reflection of the left traveling wave.

Figures 15.3 and 15.4 were generated by c02s07a.m. Figure 15.3 displays a triangular wave decomposing into its right and left traveling components. This was demonstrated in the previous chapter. As the animation progresses, the traveling waves reach the end points of the string, as in the case of Figure 15.4. The left boundary is free and the right boundary is fixed.

The wave at the right boundary will eventually reach an amplitude of twice its original height and return towards the center of the string on the same side (upper) as it started. The wave at the left boundary will pass through zero amplitude and become inverted before returning to the center of the string. Once at the center, the two waves will cancel each other out and then continue to the opposite boundaries. After another reflection, the initial shape of the string is restored for an instant. The time elapsed until the initial shape is restored is the time it takes the wave to travel twice the length of the string (2L) or T = 2L/c. This process will repeat itself indefinitely, because no attenuation

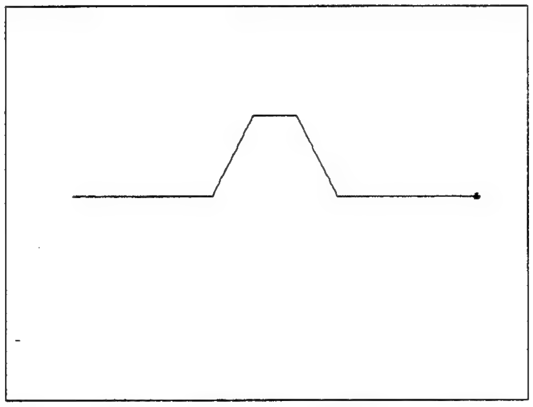
Figure 15.3 Transverse Displacement on a String Breaking into Traveling Waves

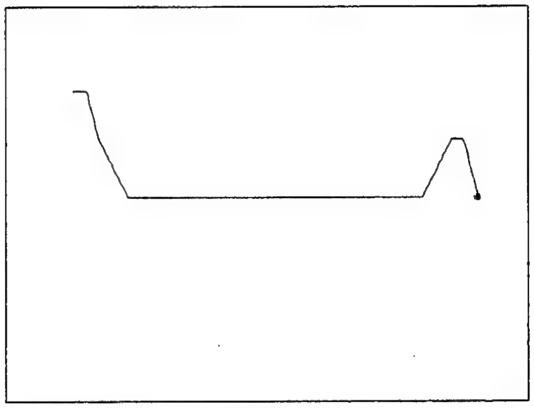
Figure 15.4 Traveling Waves at on a String at Free and Fixed Boundaries

processes are involved.

There are really no differences between the methods of c02s07.m and c02s07a.m. They both show the interaction of traveling waves. In the case of the latter program, we simply do not display the reflection of the waves approaching the boundaries as we did in the former. C02s07b.m was developed to confirm this approach. In this program, the Fourier series solution was determined (as derived in standard mathematical treatises on Fourier's theorem) for the specific case of free boundary conditions. The solution was animated by plotting the summation of thirty harmonic components over a range of time increments. This produces motion identical to that displayed by first two programs when set up for the free boundary case.

B. ALGORITHM LIMITATIONS

C02s07.m and c02s07a.m are both limited to cases where the mechanical impedance at the string ends is either zero or infinite. In other words, the strings ends must either be completely free or rigidly fixed, nothing between.

As indicated above, c02s07b.m only displays string motion for the free-free case, but it can be readily modified to display the same cases the first two are capable of by solving the Fourier series for the appropriate boundary conditions.

C. SALIENT FEATURES OF c02s07.m AND c02s07a.m

These two programs are almost identical and vary mostly in the way they present their displays and the fact that the latter program deals with two boundary points.

1. Boundary Conditions

Boundary conditions are modified through the variables “BC” for the case of c02s07.m and “leftBC” and “rightBC” for c02s07b.m. Setting the boundary condition equal to negative one generates the rigid (odd symmetry) condition and setting it to positive one generates the free (even symmetry) condition.

2. Initial Wave Shape

The initial wave shape of a freely vibrating string is determined by the method in which it was generated, such as by striking, plucking or bowing (KFCS p.34). To modify the initial wave shape of these simulations, change the vector “shape” to mimic the shape of your choosing. You only need to generate the shape of the initial wave. The program will take care of the reflections. Wave shapes for a right triangle, an isosceles triangle and a semi-circle are provided in the coding.

3. Display Rate

The display rate of animation can be modified through the variable "pace". Setting "pace" to a value between 0 and 1 will give the fastest display rate. This essentially negates the pause command. Setting "pace" to a value of 1 or greater will cause the display to pause for a time interval equal to the number of seconds entered in "pace".

D. SALIENT FEATURES OF c02s07b.m

1. Number of Harmonic Components

The number of harmonic components added to create the displacement of the string is determined by variable "n". To create an image that is closer to the predicted displacement, increase the number of harmonic components. For example, "for n=1:100" will combine a hundred components to create a sharper image, but the program will run slower.

2. Time Increments

To cause the animation to run longer or in smaller increments modify the "t" vector. For example, "t=linspace(0,10,200)" will cause the animation to run twice as long.

E. ALGORITHM c02s07.m

```
% c02s07.m  reflection at a boundary

% animated plot of the interaction of a triangular wave and
% its reflection approaching a rigid or free boundary

clear, figure(1), clf

shape=[1:50];                                 % right triangle

%shape=[1:25,25:-1:1];                        % isosceles triangle

%shape=sin(linspace(0,pi,50));                % semi-circle

BC=-1;                                        % boundary condition: -1=rigid, +1=free

pace=.1;                                      % .1 for fast, 1 for slow display

right=[zeros(1,100),shape,zeros(1,21)];

left=fliplr(-right);

leftright=left+right;

len=length(leftright);

L=line(1:len/2,leftright(1:len/2),'erasemode','xor','linestyle',':');

R=line(len/2+1:len,leftright(len/2+1:len),'erasemode','xor','linestyle','-');

wall=line([len/2,len/2],[-2*max(shape),2*max(shape)],'linestyle','- -','color',[0 1 0]);

axis([0 len -2*max(shape) 2*max(shape)])

axis('off')

if BC==-1

  line(len/2,0,'linestyle','.','markersize',25,'color',[0 .7 0]);

end

for loop=1:len-90;

  right(1)=[]; right(len)=0;
```

```
 left=fliplr(BC*right);
 leftright=left+right;
 set(L,'ydata',leftright(1:len/2))
 set(R,'ydata',leftright(len/2+1:len))
 pause(pace);
end
```

F. ALGORITHM c02s07a.m

```
% c02s07a.m Boundary Conditions
% animated plot of transverse waves traveling on a string
% with rigid or free boundary conditions at the string ends
clear, figure(1), clf
leftBC=+1;     % boundary condition: -1=rigid, +1=free
rightBC=-1;    % boundary condition: -1=rigid, +1=free
shape=[1:25,25:-1:1]./25./2;
shape2=[zeros(1,100),shape,zeros(1,100)];
left=[rightBC*shape2,rightBC*leftBC*shape2,leftBC*shape2,shape2];
right=[shape2,rightBC*shape2,rightBC*leftBC*shape2,leftBC*shape2];
len=length(right);
if leftBC==-1
  line(1,0,'linestyle','.','markersize',20,'color',[0 .7 0]);
end
```

```
if rightBC==-1
   line(250,0,'linestyle','.','markersize',20,'color',[0 .7 0]);
end
both=left(751:1000)+right(1:250);
bothline=line(1:250,both);
for n=1:1000
  both=left(751:1000)+right(1:250);
  set(bothline,'ydata',both)
  axis([1 250 -1 1]), axis('off')
  left=[left(len),left(1:len-1)];
  right=[right(2:len),right(1)];
  drawnow
end
```

G. ALGORITHM c02s07b.m

```
% c02s05b.m  Boundary Conditions
% animation of Fourier series solution of the transverse wave equation
% with free end conditions
clear, figure(1), clf
x=linspace(0,4,100);          % length of string
for t=linspace(0,10,100);     % time
  sum=0;
```

```
 for n=1:30;                    % number of harmonic components
  An=1/2*(4/n/pi)^2*(2*cos(n*pi/2)-cos(n*pi/4)-cos(3*n*pi/4));
  y=An*cos(n*pi*t/4).*cos(n*pi*x/4);
  sum=sum+y;
 end
 plot(x,sum+1/4), axis([0 max(x) -1 1]), axis('off')
 drawnow
end
```

XVI. FORCED VIBRATION OF AN INFINITE STRING

Algorithm c02s08.m plots the forced vibration of a segment of an infinite string as seen in both the time and spatial domains. This program and the following discussion illustrate concepts addressed in KFCS section 2.8.

A. CONCEPT

The upper plot in Figure 16.1 shows the vibrations of a string at different instants in time. The string is driven by a sinusoidal force acting transversely at the left end of the string, at length zero. The driver is represented by a circle attached to the left end of the string. The string extends to the right toward infinity; for obvious reasons, the plot only shows a portion of the string.

Since the string extends to infinity, there are no reflected waves. Since there is no

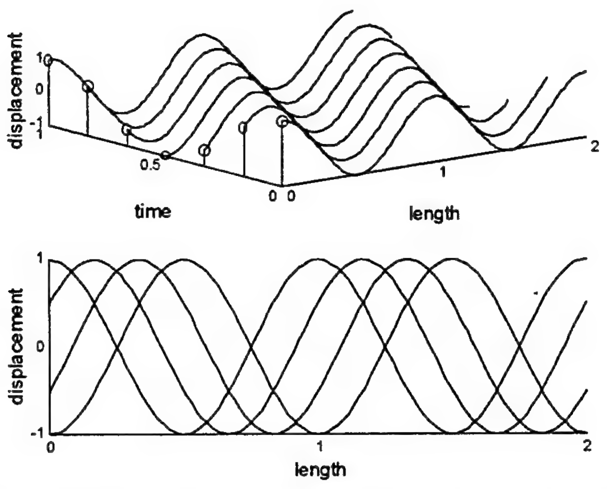

Figure 16.1 Force Vibration of an Infinite String

string extending to the left of the driver, there are also no left traveling wave components.

The lower plot shows the position of the string during the first four time instants. It was created by changing the viewing angle of the upper plot until you looked down the time axis. Its as if the time axis extends down into the paper, or screen.

B. SALIENT FEATURES OF c02s08.m

1. Frequency

The angular frequency of string is given by the variable "w", which for this plot is 2π radians per second. The driver will, therefore, go through one complete cycle or period in one second. The time axis of the upper plot shows the string progressing through one period in one second. The string can be made to go through any number of cycles by changing the "w" variable. For example, changing angular frequency to "w=5*pi" will cause the string to go through 2.5 complete cycles per second.

Looking at the positions of the driven ends of the string along the time axis, you can see that the in the time domain, the driver is indeed functioning sinusoidally. The driver follows the path of cos(wt) for one second.

2. Wavelength

The wave number, "k", determines the wavelength of the waves on the string. In this case, "k" is equal to 2π. The wavelength, determined from $2\pi/k$, is, therefore, equal to one. Wavelength is to the spatial domain as wave period is to the time domain. The lower plot show that the string goes through one complete cycle in one meter (or any desired distance unit). Note in the lower plot each of the waves have the same wavelength. As long as the driver maintains the same number of cycles per second (frequency), the waves will all have same wavelength.

To change the wavelength, change the value of the wavenumber. For example, changing to “k=4*pi”, will reduce the wavelength by half and will double the number of waves displayed in the same length of string.

C. ALGORITHM c02s08.m

```
% c02s08.m
% Forced vibration of a string of infinite length
clear, figure(1), clf
w=2*pi;                          % angular frquency
k=2*pi;                          % wave number
x=linspace(0,2,100);             % string length
t=linspace(0,1,7);               % time increments
[T,X]=meshgrid(t,x);
Z=real(exp(j*w.*T-j*k.*X));      % vertical displacement
subplot(211)
line(X,T,Z), view(-37.5,50)
line(zeros(size(t)),t,Z(1,:),'linestyle','o','color',[1 1 1])
line(zeros(2,length(t)),[t;t],[-ones(1,length(t));Z(1,:)],'color',[1 1 1])
xlabel('length'), ylabel('time'), zlabel('displacement')
subplot(212)
```

```
line(X(:,1:4),T(:,1:4),Z(:,1:4)), view([0 0])

xlabel('length'), zlabel('displacement')

set([subplot(211),subplot(212)],'xlim',[min(x) max(x)],...

 'ylim',[min(t) max(t)],'zlim',[min(min(Z)) max(max(Z))],...

 'ticklength',[0 0],'ytick',[0 .5 1],'ztick',[-1 0 1],...

 'fontsize',9,'xtick',[0:4])
```

XVII. FORCED VIBRATION OF A FIXED STRING

Algorithm c02s09a1.m displays an animated plot of the shape of a string subject to two sinusoidal waves traveling in opposite directions. Algorithm c02s09a2.m plots the shape of a driven-fixed string at various times in a single cycle. These programs and the following discussion illustrate concepts addressed in KFCS section 2.9(a).

A. CONCEPT

Consider a string of finite length, driven at one end and bounded in some form at the other. The equation of motion is satisfied by the sum of two harmonic waves traveling in opposite directions. Figure 17.1 shows one frame of the animation created by c02s09a1.m. It depicts the sum of two sinusoids, the upper one traveling to the right and the middle traveling to the left. The bottom wave is the sum of the upper two.

In this case, both the left and right traveling waves have the same amplitude and

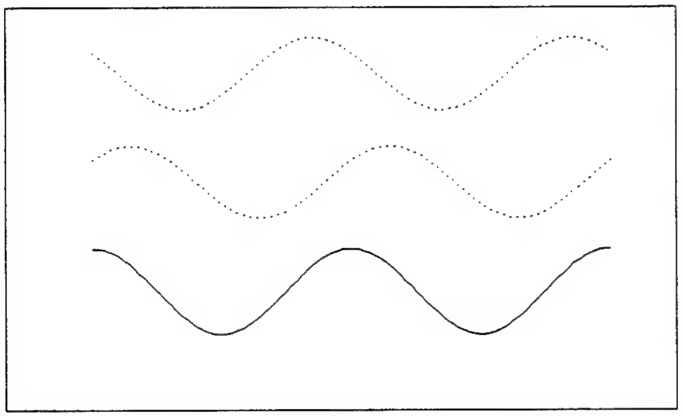
Figure 17.1 Standing Wave as Sum of Two Sinusoids

frequency. When combined, they form a wave that propagates neither to the right nor left, referred to as a standing wave.

Figure 17.2 shows consecutive positions of a string driven at the left and rigidly terminated at the right. Note the presence of a standing wave.

Important characteristics of standing waves are readily observed in this plot: 1) The position on the string where the displacement is always zero at nodes. 2) The positions of maximum displacement are antinodes. 3) The moving portions of the string

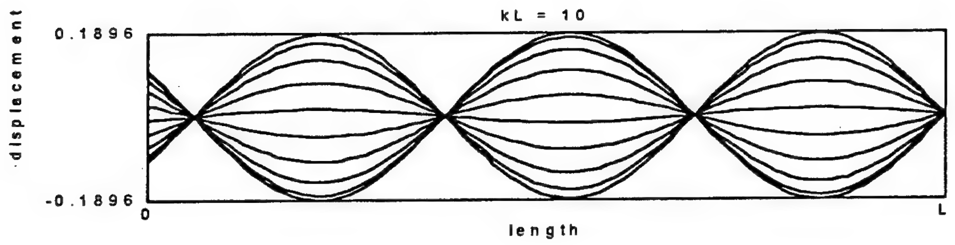

Figure 17.2 Standing Wave on a Rigidly Fixed String

between nodes are loops.

B. SALIENT FEATURES OF c02s09a1.m

1. Standing Wave Requirements

Standing waves are not developed unless the frequency and amplitude are the same for both component waves. Observe the results of changing either of these components. For example, changing vector "w1" to "w1=4*pi" will double the frequency of the right traveling (top) wave. When this occurs, the resulting (bottom) wave will no longer maintain its position but will propagate in the direction of the component with the highest frequency , in this case to the right.

If a component has the same frequency but the amplitude is changed, a standing wave will result, but with a minimum in place of the nodes. For example, changing constant "A2" to "A2=2" to double the amplitude of the left traveling (middle) wave results in a standing wave ratio (ratio of maximum to minimum amplitude) of 3/1.

C. SALIENT FEATURES OF c02s09a2.m

1. Node Migration

Varying the relationship between wavelength and string length will vary the position of the standing wave nodes. This relationship is described by the variable "kL", which represents the multiplication of the wave number, "k" and string length, "L".

Changing kL to an odd integer multiples of $\pi/2$, such as "kL=3*pi/2", will cause an antinode to appear at the driver position. Note the extremely large amplitudes of the string. When an ideal string is driven at an antinode, it will reach an infinitely large maximum amplitude. While MATLAB can't plot to infinity, it does plot to a rather large amplitude, around ten to the fifteenth.

Changing kL to integer multiples of π, such as "kL=4*pi", will result in an antinode at the driver position. Note the extremely small maximum amplitudes obtained by the string. The maximum amplitude of the string is equal to that of the driver. As the value of kL for an integer multiple of π increases to the next highest odd integer multiple of π/2, the node at the driver moves away from driver, and the maximum amplitude of displacement dramatically increases.

D. ALGORITHM c02s09a1.m

```
% c02s09a1.m   Forced-Fixed String
% Standing wave as sum of two traveling waves
clear, figure(1), clg
w1=2*pi;                                           % y1 freq
w2=2*pi;                                           % y2 freq
A1=1;                                              % y1 amplitude
A2=1;                                              % y2 amplitude
k1=2*pi;                                           % y1 wave number
k2=2*pi;                                           % y2 wave number
x=linspace(0,2,100);                               % length
t=linspace(0,4.1,200);                             % time
top=line(x,A1*sin(-k1*x)+2*A1+3*A2,'linestyle',':');    % y1 line
mid=line(x,A2*sin( k2*x)+1*A1+2*A2,'linestyle',':');    % y2 line
bot=line(x,A1*sin(-k1*x)+A2*sin(k2*x));            % y line
axis([0 max(x) -A1-A2 3*A1+3*A2]), axis('off')
[T,X]=meshgrid(t,x);
y1=A1*sin(w1*T-k1*X);                              % right traveling wave (top)
y2=A2*sin(w2*T+k2*X);                              % left traveling wave
```

```
(middle)
y=y1+y2;                                        % standing wave (bottom)
y1=y1+2*A1+3*A2; y2=y2+A1+2*A2;
for n=1:length(t)
  set(top,'ydata', y1(:,n));
  set(mid,'ydata', y2(:,n));
  set(bot,'ydata', y(:,n));
  drawnow
end
```

E. ALGORITHM c02s09a2.m

```
% c02s09a2.m
% Forced vibration of a finite string
clear, figure(1), clf
kL=10;
w=2*pi;                                         % angular frquency
k=2*pi;                                         % wave number
L=kL/k;                                         % string length
x=linspace(0,L,100);                            % length increments
t=linspace(0,1,20);                             % time increments
[T,X]=meshgrid(t,x);
Z=sin(k*(L-X)).*exp(j*w*T)/cos(kL)/k;
```

```
Z=real(Z);

axes('position',[.15 .3 .75 .4]), plot(x,Z)

title(['kL = ',num2str(kL)])

xlabel('length'), ylabel('displacement')

axis([min(x) max(x) min(min(Z)) max(max(Z))])

set(gca,'xtick',[0,L],'xticklabels',['0';'L'],'ytick',...
  [min(min(Z)) max(max(Z)) -1 1],'ticklength',[0 0])
```

XVIII. FORCED VIBRATION OF A MASS-LOADED STRING

Algorithm c02s09b1.m computes the resonance frequencies of a forced, mass-loaded string. Algorithm c02s09b2.m plots the envelope of the transverse displacement of this string. These programs and the following discussion illustrate concepts addressed in KFCS Section 2.9(b).

A. CONCEPT

The motion of a string, driven at one end and attached to a mass free to move transversely at the other, can be very different from that of a rigidly fixed string. As discussed in the previous chapter, the standing wave of a driven-fixed string has nodes whose locations are fairly easy to determine. Additionally, there is always a node at the string's fixed end. Only when the termination of a mass-loaded string is very heavy will it not share these characteristics.

The resonant frequencies of a mass-loaded string are determined by solving the transcendental equation, $tan(kL) = - (m/m_s)kL$, KFCS equation 2.30, where m is the termination mass and m_s is the string's mass. The values of kL which are the roots of this equation determine the resonant frequencies of the system.

Figure 18.1, produced by c02s09b1.m, shows a graphical means of finding the resonant kL values. The intersections of the line, $-(m/m_s)kL$, with the tangent lines of kL are the roots of the equation. The algorithm also determines a much more exact solution

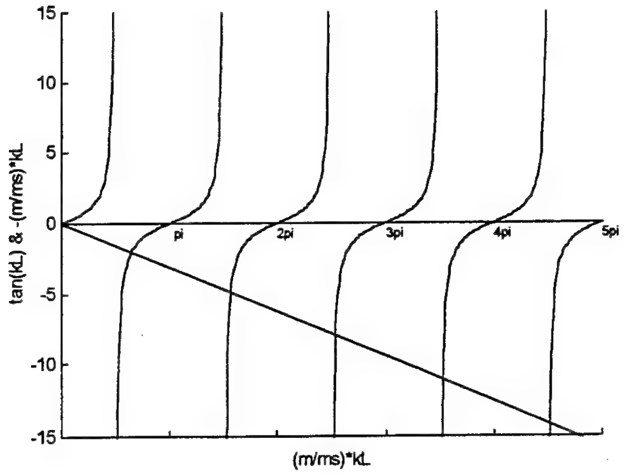

Figure 18.1 Graphical Means of Determining Mass-Loaded String's Resonant Frequencies

and displays the results in the MATLAB command window. For example, the algorithm computed the roots of the system illustrated in the first figure to be 2.0288, 4.9124, 7.9540, 11.0856, and 14.2072.

Figure 18.2 depicts the envelope of the transverse motion of a mass-loaded string driven at frequencies other than resonance. In other words, these graphs indicate the maximum displacement of string along its length; all steady state motion of the string will occur within this envelope. This figure was produced by algorithm c02s09b2.m.

B. ALGORITHM LIMITATIONS

While algorithm c02s09b1.m is easy to use and fairly quick, the roots obtained are not exact solutions. Their accuracy is determined by the variable "decimal". For example, "decimal=4", will compute roots to an accuracy of four decimal places after the integer

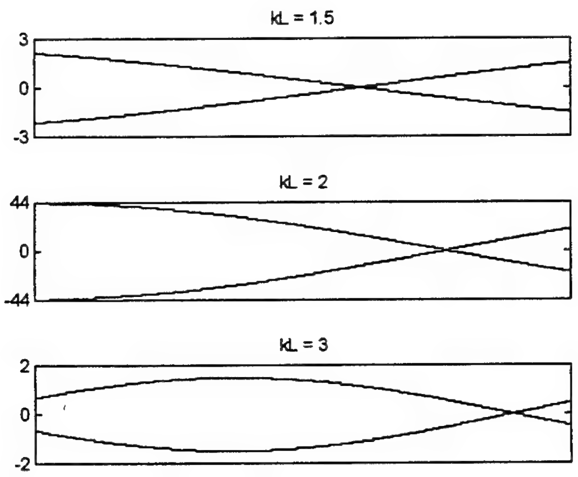

Figure 18.2 Envelope of Mass-Loaded String Motion

and will compute a dozen such roots in less than a second on an 80486 processor. "decimal=6" will provide an accuracy to the sixth decimal place, but it will take almost a full minute to compute one root.

C. SALIENT FEATURES OF c02s09b1.m

1. Harmonic Number

The quantity of resonant frequencies and their harmonic numbers are determined by the variable, "Nroots". To find the first five resonant frequencies, for example, the variable should be set to "Nroots=1:5". To find the first, fifth and tenth harmonics only, the variable should be set to "Nroots=[1,5,10]"

Notice that due to the speed of tangent approach to $\pi/2$, $3\pi/2$, and $5\pi/2$, the roots are much more evenly spaced at higher values of kL. At high frequencies, a mass-loaded string appears to have a rigid termination.

2. Termination and String Mass Ratio

The variable "m" is the ratio of mass of the termination mass to mass of the string. For example, "m=1", means that the termination mass is equal to the mass of the string. This case is plotted in the first figure and in KFCS Figure 2.10. When the termination mass is much greater that the string mass, such as when "m=100", then the termination approximates that of a rigid end. When the termination mass is very small, "m=.01", then the termination approaches that of a free end.

D. SALIENT FEATURES OF c02s09b1.m and c02s09b2.m

1. Behavior Near Resonance

From algorithm c02s09b1.m, it was determined that, when termination mass is equal to the string mass , "m=1", the fundamental is 2.03. In the Figure 18.2, "m=1", but kL varies between 1.5 and 3. Notice the amplitude of the driver at the left of the figure. With kL=2, the amplitude is much larger than that of the other two plots. Since at kL=20, the string is driven very near resonance. The top and bottom plots are on either side of resonance and show a much smaller amplitude of the driver.

The driving frequency of each plot is determined by the vector "kL". From algorithm c02s09b1.m, it was determined that the second harmonic is at kL=4.91. To plot the envelope for frequencies near the second resonance, you could use "kL=[4.5,5,6]".

In Figure 18.3, the string is driven at frequencies near the first, second and third harmonics (kL = 2.03, 4.91, and 7.98). The string driven near the first harmonic has one node, the second has two and the third three. Notice that the closer the string is driven to

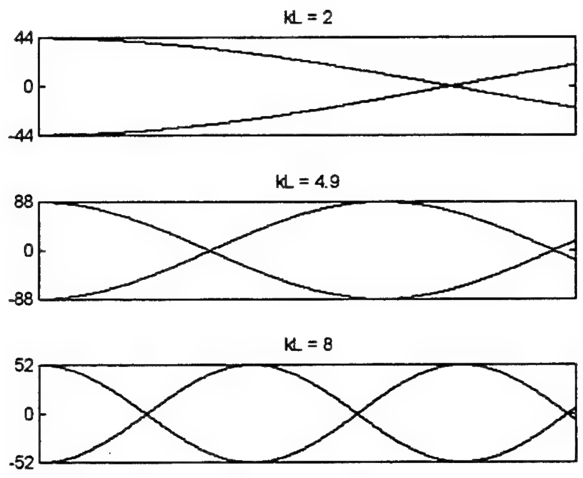

Figure 18.3 Mass-Loaded String Driven Near First, Second, and Third Harmonics

one of its resonant frequencies, the larger the amplitudes of the driver.

E. ALGORITHM c02s09b1.m

```
% c02s09b1.m
% finds resonant frequencies of mass loaded string
clear, figure(1), clf
m=1;                                % m/ms
Nroots=1:5;                         % roots (resonant frequencies) to find
decimal=4;                          % accuracy (6 or greater is slow to compute)
root=[]; kLv=[];
for r=Nroots;
  kL=[(2*r-1)*pi/2+eps:.1:r*pi];
```

```
 [val,pos]=min(abs(m*kL+tan(kL)));
 for d=1:decimal;
  kL=[kL(min(1,pos-1)):10^(-d):kL(min(pos,pos+1))];
  [val,pos]=min(abs(m*kL+tan(kL)));
 end
 root=[root,kL(pos)];
end
root
for r=0:max(Nroots),                    % tan plot
 tv=linspace((2*r-1)*pi/2+.01,(2*r+1)*pi/2-.01,100);
 line(tv,tan(tv))
end
line([0,10e5],[0,-m*10e5]);             % kL line
line([0,max(Nroots)*pi],[0,0],'linestyle',':');
text([1:5]*pi,-.8*ones(1,5), [' pi';'2pi';'3pi';'4pi';'5pi'],'fontsize',[9])
set(gca,'ytick',[-15:5:15],'ygrid','off', 'xtick',[0:6]*pi,'xticklabels',[])
xlabel('(m/ms)*kL'), ylabel('tan(kL) & -(m/ms)*kL')
axis([0,5*pi,-15,15])
```

F. ALGORITHM c02s09b2.m

```
% c02s09b2.m  Forced, Mass-Loaded Sting
% Plots envelope of string for various driving freqs
```

```
clear, figure(1), clf

kL=[1.5,2,3];                    % kL near fundamental

kL=[4.5,5,6];                    % kL near 2nd harmonic

kL=[2,4.9,8];                    % kL near 1st, 2nd, 3rd harmonics

m=1;                             % m = m/ms

x=linspace(0,1,1000);            % x = x/L

for n=1:length(kL);

 subplot(length(kL),1,n)

 na=(1+j*m*kL(n))/(m*kL(n)*cos(kL(n))+sin(kL(n)));

 nb=(1-j*m*kL(n))/(m*kL(n)*cos(kL(n))+sin(kL(n)));

 A=na*exp(j*kL(n));

 B=nb*exp(-j*kL(n));

 y=real(A/kL(n)*exp(j*(-kL(n)*x))+B*exp(j*(kL(n)*x)));

 plot(x,y,'y',x,-y,'y')

 top=ceil(max(abs(y)));

 axis([0,1,-top,top])

 title(['kL = ',num2str(kL(n))])

 set(gca,'ytick',[-top,0,top],'xtick',[])

end
```

XIX. LONGITUDINAL TRAVELING WAVES

Algorithm c03s03.m produces an animated simulation of particles in a bar effected by the passage of longitudinal traveling waves. It also graphs the longitudinal displacement and particle density. This program and the following discussion illustrate concepts addressed in KFCS Section 3.3.

A. CONCEPT

Figure 19.1 is one frame of an animation of a longitudinal traveling wave. The circles mimic the motion of particles within a bar. The bar is infinitely long so that boundary reflection can be discounted.

The dots within the particle circles mark the center of the particles when they are at rest. Notice that when the displacement graph crosses zero displacement, the particles

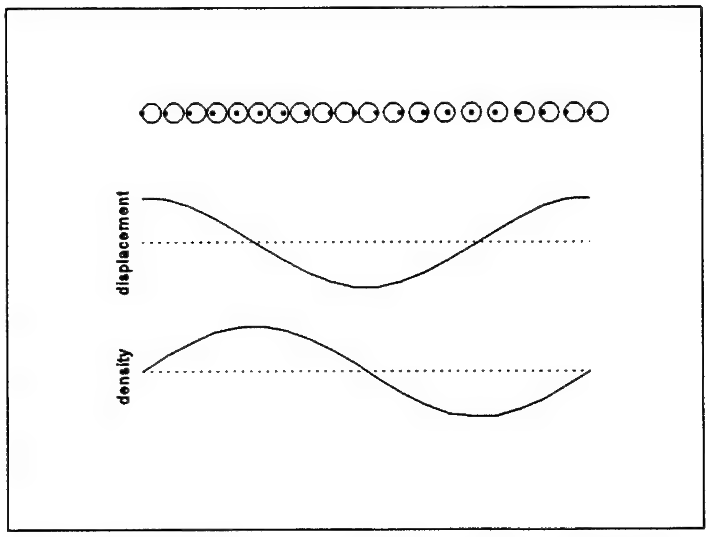

Figure 19.1 Longitudinal traveling waves in a bar

at their resting position. The particles are more closely packed when density is the greatest. When the particles are furthest apart, the density is least. Increasing density (positive slope on the density graph) indicates the particles are undergoing compression, while decreasing density (negative slope) indicates tension.

B. SALIENT FEATURES OF c03s03.m

1. Wave Number

The wave number of bar in Figure 19.1 is 2π. Since the figure displays one unit length of bar, the longitudinal wave will go through one complete tension-compression cycle in the length of bar illustrated. Remember, this bar is considered to be of infinite length.

To display more than one cycle, change the variable "k". For example, "k=6*pi", will animate three complete cycles.

2. Display Speed and Duration

To increase the speed with which the waves cross the screen, increase the driving frequency, "w". The display can be paused within each time increment by changing the "pause" command line near the bottom of the algorithm. For example, "pause(2)" would cause a two second pause between each time increment.

Any value of pause less than unity essentially negates the "pause" command and causes the screen to redraw with the new position data. For example, since "pause" is currently set to 0.1 seconds (less than unity), the screen immediately redraws without pausing between time increments. This performs the same function as MATLAB's "drawnow" command.

To pause indefinitely between each increment, change the pause command to simply "pause". The display will wait until you press a key before continuing. This can become tiresome because you can't resume animated display. To terminate the pause command (and the algorithm) press the "control" key and the "c" key together.

C. ALGORITHM c03s03.m

```
% c03s03.m
% animated plot of longitudinal traveling waves in a bar
clear, figure(1), clf, set(gcf,'backingstore','off')
k=2*pi;             % wave number
w=1;                % angular freq
N=20;               % number of particles
x=linspace(0,1,N);  % initial particle x-positions
y=zeros(size(x));   % initial particle y-positions
line([0,max(x)],[-2,-2],'linestyle',':','color',[1,1,1],...
 'erasemode','background');
line([0,max(x)],[-4,-4],'linestyle',':','color',[1,1,1],...
 'erasemode','background');
axis([-.1,1.1,-5.5,1]), axis('off')
text(-.05,-2.8,'displacement','rotation',90,...
 'erasemode','background')
```

```
text(-.05,-4.5,'density','rotation',90,...
 'erasemode','background')
line(x,y,'linestyle','.','markersize',9,'color',[1,1,1],...
 'erasemode','background');
part_line=line(x+.03*sin(k*x),y,'linestyle','o',...
 'erasemode','xor','markersize',10.5);
disp_line=line(x,.7*sin(k*x)-2,'erasemode','xor');
rho_line=line(x,-.7*cos(k*x)-4,'erasemode','xor');
for t=linspace(0,3*pi/2,50);   % time incrementation
  disp=sin(k*x-w*t);           % displacement (longitudinal)
  rho=-cos(k*x-w*t);           % density
  part=x+.02*disp;             % particle position
  set(part_line,'xdata',part)
  set(disp_line,'ydata',.7*disp-2)
  set(rho_line, 'ydata',.7*rho-4)
  pause(.1)
end
```

XX. LONGITUDINAL STANDING WAVES

Algorithm c03s04.m produces an animation depicting longitudinal standing waves in a bar with fixed end conditions. Algorithm c03s04a.m traces the motions of the bar's particles over time. These programs and the following discussion illustrate concepts addressed in KFCS section 3.4.

A. CONCEPT

Figure 20.1 shows one frame of an animation similar to that produced in the previous chapter with algorithm c03s03.m. The difference being that this figure depicts longitudinal motion for a bar fixed at both ends with the nodal positions marked by dotted vertical lines. Notice that as the animation progresses, the nodal points remain stationary. In other words, this is a standing wave.

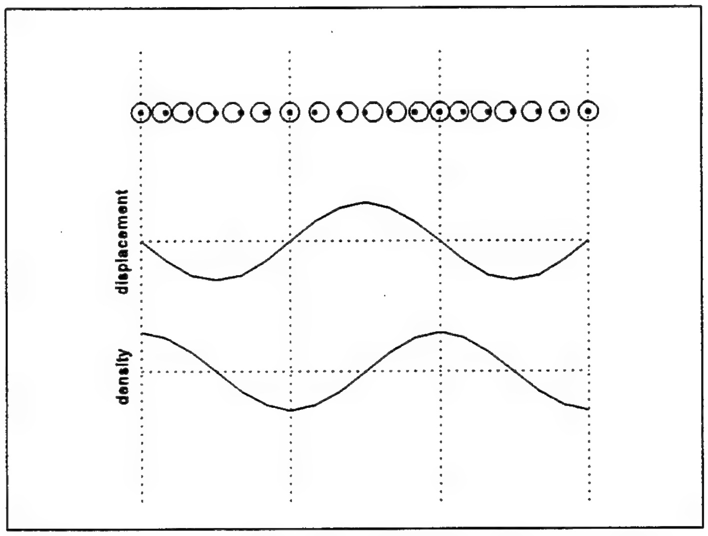

Figure 20.1 Longitudinal Standing Waves in a Bar Clamped at Both Ends

The circles show the instantaneous position of the particles and the dots within the circles mark the positions of the particles at rest. Notice how the particles at the nodal position remain fixed in place. If the bar were clamped at these nodal positions, the motion would not be effected.

Figure 20.2 shows a time trace of the motion of the bar's particles. Nodal positions are marked with the symbol "^". Notice that there is no longitudinal motion of particles at nodes.

B. ALGORITHM LIMITATIONS

It is possible to input wave numbers other than those of the allowed modes of vibration. In this situation, motion of the system is still plotted, but fails to meet the required fixed end conditions. Accurate simulation of the appropriate motion is not produced at other than allowed frequencies.

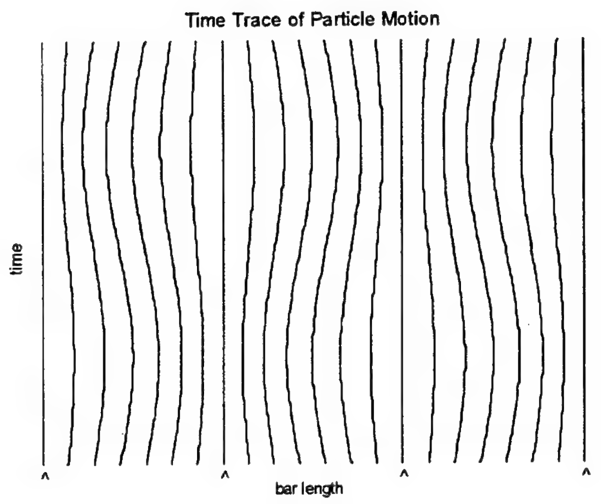

Figure 20.2 Time Trace of Particles in a Bar Clamped at Both Ends

C. SALIENT FEATURES OF c03s04.m

1. Mode Number

The wave number corresponding to the allowed modes of vibration is contained in the variable, "k". Since the bar is of unit length, the formula for determining the allowed modes is $k=n\pi$ (KFCS eq 3.12) where n is the mode number and is any positive integer. To animate the motion of the second mode, for example, use "k=2*pi". The algorithm will automatically mark the nodal positions. Nodal positions will only be marked when driving at allowed frequencies.

2. Display Pace

The "pause" command line can be modified as discussed in the previous chapter to change the animation display rate.

D. SALIENT FEATURES OF c03s04a.m

1. Node Number

Mode numbers in this algorithm are changed in the same manner as those in c03s04.m. Regardless of the mode number entered, approximately twenty particle positions will be traced and there will always be a particle traced and marked at the nodal position. This allows easy spotting of nodal positions, but prevents plotting at over the twentieth mode. If this poses a problem, increase the integer in the "node" command line. For example, "node=round(30*pi/k)" will plot approximately thirty particle traces.

E. ALGORITHM c03s04.m

```
% c03s04.m

% animated plot of longitudinal standing waves

% in a bar clamped at both ends (fixed-fixed)

clear, figure(1), clf, set(gcf,'backingstore','off')

k=3*pi;                          % wave number of 3rd mode

w=1;                             % angular freq

N=19;                            % number of particles

x=linspace(0,1,N);               % initial particle x-positions

y=zeros(size(x));                % initial particle y-positions

line([0,max(x)],[-2,-2],'linestyle',':','color',[1,1,1],...

 'erasemode','background');

line([0,max(x)],[-4,-4],'linestyle',':','color',[1,1,1],...

 'erasemode','background');

axis([-.1,1.1,-5.5,1]), axis('off')

text(-.05,-2.8,'displacement','rotation',90,...

 'erasemode','background')

text(-.05,-4.5,'density','rotation',90,...

 'erasemode','background')

line(x,y,'linestyle','.','markersize',9,'color',[1,1,1],...

 'erasemode','background');
```

```
if k/pi==round(k/pi)
 line([pi/k,pi/k]'*[0:5],[-6,1]'*ones(1,6),'linestyle',...
 ':','color',[1,1,1])
end
part_line=line(x+.03*sin(k*x),y,'linestyle','o',...
 'erasemode','xor','markersize',10.5);
disp_line=line(x,.6*sin(k*x)-2,'erasemode','xor');
rho_line=line(x,-.6*cos(k*x)-4,'erasemode','xor');
for t=linspace(0,3*pi/2,50);          % time incrementation
 disp=sin(k*x)*sin(w*t);              % displacement (longitudinal)
 rho=-cos(k*x)*sin(w*t);              % density
 part=x+.02*disp;                     % particle position
 set(part_line,'xdata',part)
 set(disp_line,'ydata',.6*disp-2)
 set(rho_line, 'ydata',.6*rho-4)
 pause(.1)
end
```

F. ALGORITHM c03s04a.m

```
% c03s04a.m
% longitudinal standing waves in a fixed-fixed bar
% time trace of particle positions
```

```
clear, figure(1), clf
k=3*pi;                                 % wave number of 3rd mode
w=2*pi;                                 % angular freq
t=linspace(0,1,100);                    % time
node=round(20*pi/k);                    % number of lines per node
N=node*k/pi+1;                          % number of lines total
x=linspace(0,1,N);                      % length incrementation
dx=x(2)-x(1);                           % line separation at rest
[X,T]=meshgrid(x,t);
trace=sin(w*T).*sin(k*X);
plot(trace./40+ones(size(t'))*x,T,'y')
axis([0,1,0,1]); axis('off')
title('Time Trace of Particle Motion','fontsize',13)
text(.43,-.06,'bar length','fontsize',11)
text(-.05,.45,'time','rotation',90,'fontsize',11)
node_mark=dx*node*[0:N/node];
text(node_mark-.005,-.03*ones(size(node_mark)),'^')
```

XXI. TRANSVERSE VIBRATION OF A CLAMPED-FREE BAR

Algorithm c03s11.m plots a graphic means of determining the normal mode frequencies of transverse vibration of a clamped-free bar. It also determines the exact solution and plots the transverse motion using these results. This program and the following discussion illustrate concepts addressed in KFCS Section 3.11.

A. CONCEPT

The solution of the equation of transverse motion for a bar clamped at one end is only satisfied for certain allowable frequencies. KFCS Equation 3.55 provides solutions based on the intersection of the cotangent and positive/negative hyperbolic tangent. The top graph in the Figure 21.1 shows a graphical means of determining these solutions. This graphs shows five intersection points. The hyperbolic tangent curves approach ±1 so

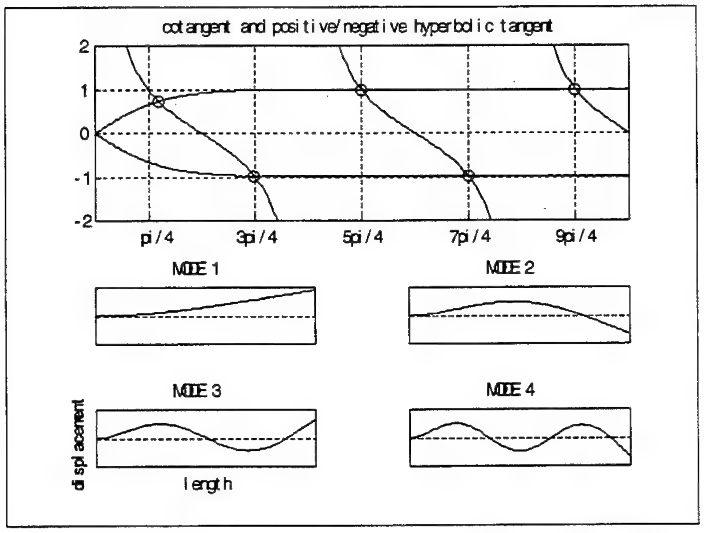

Figure 21.1 Transverse Motion of a Fixed-Free Bar

quickly that, after the first two, the intersections become very close to odd intervals of $\pi/4$.

Algorithm c03s11.m shows the $\pi/4$ multiples of the first twelve intersection points with accuracy to the fifteenth decimal point. These intersections are provided below:

1.19372832538893	13.00000000172371
2.98835122854670	14.99999999992552
5.00049389233340	17.00000000000322
6.99997863969489	18.99999999999986
9.00000092303017	21.00000000000001
10.99999996011219	23.00000000000000

Notice how quickly the intersections approach integer multiples of $\pi/4$.

The four smaller graphs in the figure plot the transverse motion of the bar for the first few modes. These graphs are also illustrated in KFCS Figure 3.8.

B. ALGORITHM LIMITATIONS

High decimal accuracy is quickly obtained because of the solution regularity after the first two modes. Rough solutions are provided to the algorithm for the first two modes so that processing time can be reduced. The algorithm begins at the rough solutions and refines them to the fifteenth decimal place.

C. SALIENT FEATURES OF c03s11.m

1. Transcendental Equation Roots

To determine additional roots, beyond the twelve provided, change the vector "Nroots" to indicate the roots desired. For example, "Nroots=14:20" will provide the fourteenth through twentieth roots.

2. Free-Free Bar

The roots of the fixed-free bar were found by the intersections of cot(x)=±tanh(x), similarly, the roots of the free-free bar are found by the intersections of tan(x)=±tanh(x). To determine these roots with algorithm c03s11.m simply modify the fourteenth code line by replacing "cot" with "tan" and modify the rough solutions of the first two modes accordingly: "if r==1; rtemp=3.0112*pi/4" and delete the following line which references "r==2".

D. ALGORITHM c03s11.m

```
% c03s11.m  transverse motion of a bar clamped at one end
% plots graphic means of determining allowable frequencies,
% determines exact solutions, and plots transverse motion
clear, figure(1), clf
Nroots=1:12;            % roots (allowable frequencies) to find
root=[];  x=[];
for r=Nroots;
 if r==1; rtemp=pi/4*1.194;
   elseif r==2; rtemp=pi/4*2.988;
   else rtemp=(2*r-1)*pi/4;
 end
 for n=[1:4]*(-3)
   x=linspace(rtemp-10^n,rtemp+10^n,1e3);
   [val,pos]=min(abs(abs(cot(x))-tanh(x)));
   rtemp=x(pos);
  end
  root=[root,x(pos)];
end
format long
root'/pi*4
format
subplot(211),                %%%%%% plot of cot and +-tanh
x=linspace(acot(2),pi-acot(2),100);
xx=linspace(0,2.5*pi,300);
plot(x,cot(x),'y',x+pi,cot(x+pi),'y',x(1:50)+2*pi,...
  cot(x(1:50)+2*pi),'y',xx,tanh(xx),'y',xx,-tanh(xx),'y',...
  root(1:min(5,length(root))),cot(root(1:min(5,length(root)))),'go')
axis([0,2.5*pi,-2,2]), grid
set(gca,'ytick',[-2,-1,0,1,2],'xtick',[1:2:9]*pi/4,...
  'xticklabels',[' pi/4';'3pi/4';'5pi/4';'7pi/4';'9pi/4'])
title('cotangent and positive/negative hyperbolic tangent')
subplot(212),                %%%%%% plot of transverser motion
x=linspace(0,1,101);
for n=1:4
```

```
 N=root(n)*2;
 A=-(sinh(N)+sin(N))/(cosh(N)+cos(N));
 y=A*(cosh(N*x)-cos(N*x))+(sinh(N*x)-sin(N*x));
 subplot(4,2,4+n)
 plot(x,-y),grid,axis([0,1,-3,3])
 set(gca,'xtick',[],'ytick',0,'yticklabels',[])
 title(['MODE ',num2str(n)])
end
subplot(4,2,7)
xlabel('length'), ylabel('displacement')
```

XXII. FREE VIBRATIONS OF A RECTANGULAR MEMBRANE WITH FIXED RIM

Algorithm c04s03.m and c04s03a.m show the behavior of the normal modes of a rectangular membrane. These programs and the following discussion illustrate concepts addressed in KFCS section 4.3.

A. CONCEPT

An ideal membrane, fixed to a rectangular rim, will have normal modes as shown in the Figure 22.1, created by c04s03.m. This two dimensional motion has many similarities to the one dimensional motion of a string, fixed at both ends.

The wave number of a membrane is the combination of both a horizontal and vertical component. These discrete sets of values determine the allowed frequencies of

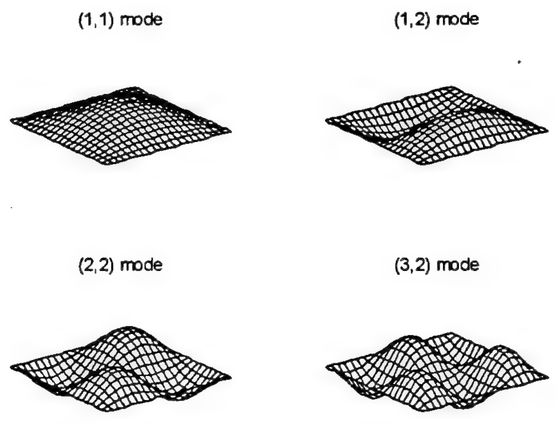

Figure 22.1 Normal Modes of a Rectangular Membrane

free vibration, and are described by the ordered pair (m,n). KFCS Equation 4.16 shows that k_x=nπ/L_x and k_y=nπ/L_y, where m and n take on integer values greater than zero, and L_x and L_y are the dimensions of the membrane. This mode descriptor (m,n) also indicate the number of nodal lines, or lines of no displacement. As shown in the figure, the (1,1) mode only contains nodal lines at the boundaries. The (3,2) mode contains two lines perpendicular to the x-direction and one line perpendicular to the y-direction, in the interior, as well as nodal lines around the rim.

The membrane between the nodes oscillates between the maximum positive and negative displacement, but with 180 degree phase difference across the nodes. c04s03a.m displays a movie that shows this motion.

B. ALGORITHM LIMITATIONS

Algorithm c04s03a.m develops a movie showing normal mode time behavior. Unfortunately, the movie function in MATLAB requires lots of random access memory, RAM. The movie may not load at all, may not run smoothly or may not show the appropriate colors if there is not enough RAM available. 15 megabytes is sufficient for smooth operation.

C. SALIENT FEATURES OF c04s03.m

1. Mode Numbers

The mode numbers, m and n, may be changed through the vectors "m" and "n". To plot the (4,6) mode, for example, the variables should be changed to read "n=4; m=6;". To display several modes at the same time, any number of mode numbers can be contained in the vectors long as "m" and "n" are the same length. The algorithm will automatically determine the appropriate number and arrangement of subplots.

2. Rim Size

The dimensions of the membrane rim are contained in the vectors "Lx" and "Ly". When Lx = Ly, the rim is a square. When Lx is much, much greater than Ly, the graph resembles the transverse displacement of a fixed-fixed string.

3. Grid Size

The density of the mesh displayed, or grid size, is contained in the variable "g". Setting "g=40", for example, will produce a mesh of 40 vertical and 40 horizontal lines.

4. Overhead View

This algorithm contains three means of viewing a membrane. An overhead view, such as that provided by the commands "pcolor" or "contour" provides a good indication

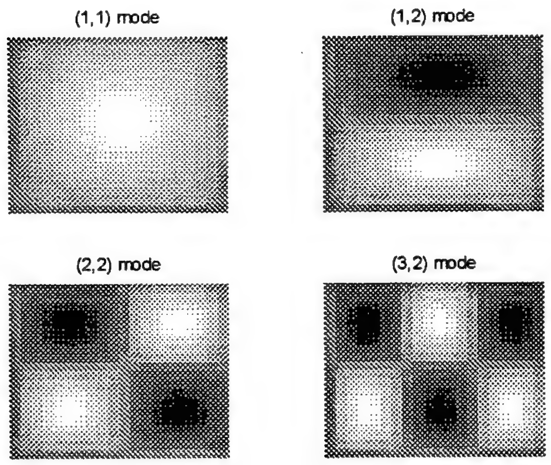

Figure 22.2 Overhead View of Rectangular Membrane Normal Modes

of nodal line position. Figure 22.2 shows a plot of the same modes shown in Figure 22.1. This plot is activated by removing the "%" symbol from the line of code beginning with the command "pcolor". A similar overhead view can be plotted by removing the "%" symbol from the "contour" command line. Be sure to negate the other two plotting means by using the "%" symbol, otherwise the graphs will be plotted using the last allowable method.

When showing overhead views, more detail can be displayed by increasing the grid size through variable "g". The grid in the Figure 22.2 is set to "g=21"

D. SALIENT FEATURES OF c04s03a.m

Mode numbers, rim dimensions, and grid size are determined just as in c04s03.m.

1. Movie

As discussed above, the MATLAB movie function requires a great deal of RAM to operate. Time behavior can also be plotted without showing a movie as shown in Figure 22.3. This was produced by changing the variable "show_movie" to any negative number. For example, setting "show_movie=-1" will cause the algorithm to graph the membrane at four different time instances. To activate the movie, set "show_movie=1".

E. ALGORITHM c04s03.m

```
% c04s03.m  normal modes of a rectangular membrane

% various views of various modes

clear, figure(1), clf

n=[1,1,2,3];                    % x-axis mode
```

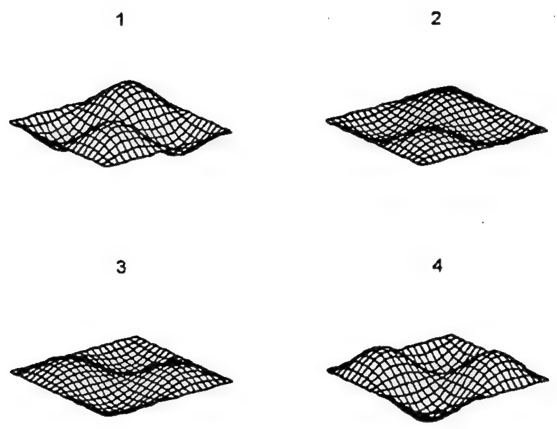

Figure 22.3 Time Behavior of Mode (2,2) Membrane

```
m=[1,2,2,2];                    % y-axis mode



Lx=1;                           % length in x direction

Ly=1;                           % length in y direction

g=21;                           % grid size

x=linspace(0,Lx,g);

y=linspace(0,Ly,g);

[X,Y]=meshgrid(x,y);

col=max(ceil(sqrt(length(n))),ceil(length(n)/2));

row=ceil(length(n)/col);

for s=1:length(n)

  kx=n(s)*pi/Lx;                % x component wave number

  ky=m(s)*pi/Ly;                % y component wave number
```

```
 Z=sin(kx*X).*sin(ky*Y);
 subplot(col,row,s)
 mesh(x,y,Z), axis([0,max(Lx,Ly),0,max(Lx,Ly),-1.5,1.5])
 %pcolor(x,y,Z), shading interp, view(2)
 %contour(x,y,Z)
 colormap('hot')
 caxis([-1,1])
 axis('off')
 title(['(',num2str(n(s)),',',num2str(m(s)),') mode'])
end
```

F. ALGORITHM c04s03.m

```
% c04s03a.m  normal modes of rectangular membrane with fixed rim
% time behavior
clear, figure(1), clf
n=2;                    % x-axis mode
m=2;                    % y-axis mode
Lx=1;                   % length in x direction
Ly=1;                   % length in y direction
show_movie=+1;          % positive=show, negative=don't
g=21;                   % grid size
kx=n*pi/Lx;             % x component wave number
```

```
ky=m*pi/Ly;                    % y component wave number
x=linspace(0,Lx,g);
y=linspace(0,Ly,g);
[X,Y]=meshgrid(x,y);
Z=sin(kx*X).*sin(ky*Y);
if show_movie>0
 N=21;                         % number of time steps
 a=sin(linspace(pi/2,5*pi/2,N));
 M=moviein(N);
 for n=1:N
  surf(x,y,Z.*a(n))
  axis([0,max(Lx,Ly),0,max(Lx,Ly),-2,2]), axis('off')
  caxis([-1,1])
  M(:,n)=getframe;
 end
 movie(M,20)
else
 a=sin([pi/2,pi/5,-pi/5,-pi/2]);
 col=max(ceil(sqrt(length(a))),ceil(length(a)/2));
 row=ceil(length(a)/col);
 for n=1:length(a)
```

```
    subplot(col,row,n)

    surf(x,y,Z*a(n))

    axis([0,max(Lx,Ly),0,max(Lx,Ly),-2,2]), axis('off')

    caxis([-1,1])

  title(num2str(n))

  end

end

colormap('hot')
```

XXIII. NORMAL MODES OF A CIRCULAR MEMBRANE AND PROPERTIES OF BESSEL FUNCTIONS

Algorithm c04s04.m plots the normal modes of a circular membrane fixed at the rim. Several kinds of Bessel functions are shown by c04s04a.m. Algorithm c04s04b.m uses function bsl.m to compute and plot the zero crossings of three orders of a Bessel function of the first kind, J_0, J_1, and J_2. These programs and the following discussion illustrate concepts addressed in KFCS section 4.4 and appendices A4 and A5.

A. CONCEPT

The Figure 23.1 was developed by c04s04.m. It shows the displacement of four modes of a circular membrane. Similar to a rectangular membrane, the modes and nodal surfaces of circular membrane are described by the ordered pair (m,n). In this case, m describes the number of radial nodal lines and n the number of azimuthal nodal circles.

The easiest way to determine a circular membrane displacement is to employ the polar coordinate system. Subsequently, the equation of motion is solved by means of functions known as Bessel functions. Many standard mathematical texts will provide a more thorough treatment of properties of Bessel functions, but the following overview should be helpful.

Bessel functions of the first kind, denoted by the letter "J_n", are commonly used

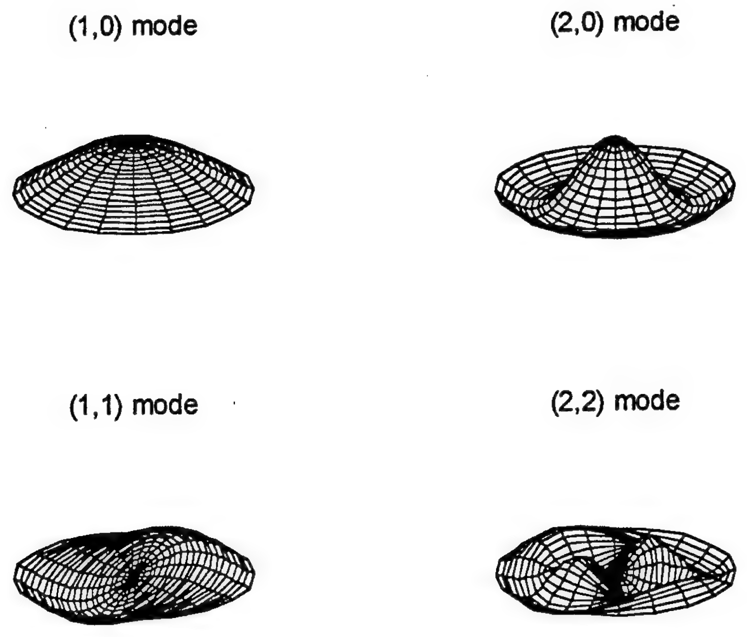

Figure 23.1 Normal Modes of a Circular Membrane

when working in polar coordinates. Bessel functions of the second kind, "Y_n", also known as Neuman functions.

Figure 23.2, produced by c04s04a.m, shows the first four orders of the Bessel functions just mentioned.

Figure 23.3 shows the zero crossing of the first three orders of a Bessel function of the first kind, J_0, J_1, and J_2. It was produced by c04s04b.m. The important things to notice are that the amplitudes of the local extrema decrease with increased argument and that the distance between zeros is not a constant. The positions of the zero crossings in

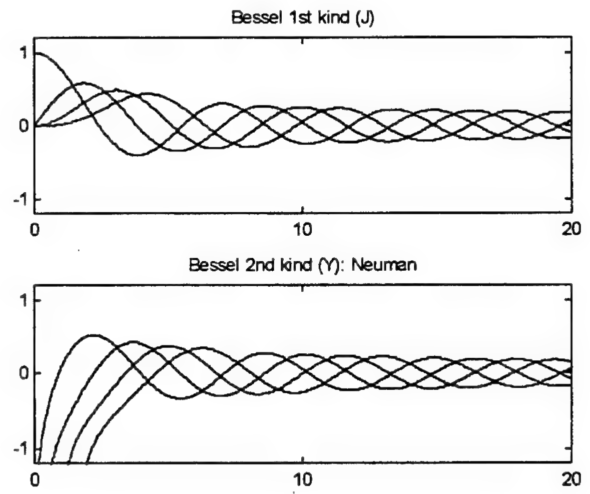

Figure 23.2 Bessel Functions of the First and Second Kind

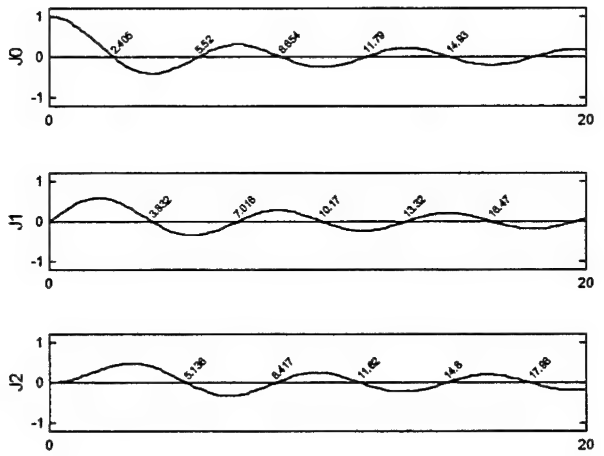

Figure 23.3 Bessel Functions of the First Kind

this plot were determined by the function bsl.m, which is described below.

B. ALGORITHM LIMITATIONS

Though the need is unlikely to arise, the algorithm for bsl.m will only produce results for the first sixty orders. If for example, the zero crossing or extrema of J_{89} are needed, the algorithm will have to be modified as described in the coding. There will be a corresponding accuracy decrease, also described in the coding.

C. SALIENT FEATURES OF c04s04.m

1. Mode Numbers

Mode numbers are contained in the vectors "m" and "n". As in the previous chapter on rectangular membranes, any number modes may be plotted and the algorithm will automatically determine an appropriate subplot arrangement.

2. Overhead View

A "pcolor" command line is included in the program to show an overhead view or transverse displacement. It can be activated by removing the preceding "%" symbol. This view doesn't print well in black and white, but presents a very useful display in color.

D. SALIENT FEATURES OF c04s04a.m

1. Bessel Orders

More, fewer, or a particular orders can be plotted by modifying the "order" vector. For example, to show just the zero order, change the vector to "order=[0]". Plotting more than three orders creates a very confusing graph.

2. Limiting Values

To see the graphs of Bessel functions over a wider range of values change the vector "x". For example, "x=linspace(0,100,1000)", will show how quickly the functions approach their limiting values and give you a good idea of how to approximate Bessel values at long distances.

E. SALIENT FEATURES OF FUNCTION bsl.m

1. Zeros and Extrema

Bsl.m is a function the provides both zero crossings and extrema or inflection points. The function's format is "[Z,E]=bsl(m,n)" where "Z" and "E" are "n" zeros and "n" extrema of a Bessel of the first kind order "m". For example, typing "[Z,E]=bsl(1,2)" at the MATLAB command window will provide the following results: "Z = 3.8317 7.0156" and "E = 1.8412 5.3315". In other words, it shows that the first two J_1 zeros are 3.8 and 7.0, and the first two extrema are 1.8 and 5.3. Typing "[Z,E]=bsl(0,5)" will provide the first row of KFCS tables A5(b) and A5(c).

Just typing "bsl(1,2)" would provide the zero crossings only.

2. Accuracy

The accuracy of this function is set to four decimal places. To increase the accuracy, and the processing time, change the upper value of the "accuracy" vector. For example, "accuracy=1:3" would provide accuracy to the seventh decimal place, but it will take 18 seconds to compute twenty extrema on a Pentium 90 processor.

3. Order Greater than Sixty

To determine the zero crossings of Bessel orders greater than sixty, increase the value of the variable "range". For example, "range=1000" will allow computations to a significantly greater order, but would the results would loose one decimal place of accuracy.

F. ALGORITHM c04s04.m

```
% c04s04.m  normal modes of circular membrane

clear, figure(1), clf

colormap('hot')

m=[0,0,1,2];                    % Bessel order

n=[1,2,1,2];                    % number of zeros or extrema

g=21;                           % grid size

x=linspace(-1,1,g);

y=x;

r=linspace(0,2,g);

theta=linspace(0,2*pi,g);

[R,Theta]=meshgrid(r,theta);

col=max(ceil(sqrt(length(m))),ceil(length(m)/2));

row=ceil(length(m)/col);

for s=1:length(m)

  subplot(col,row,s)

  [z,e]=bsl(m(s),n(s));

  jmn=z(n(s));  % nth zero of mth order Bessel of 1st kind

  kmn=jmn/max(r);

  Z=besselj(m(s),kmn*R).*cos(m(s)*Theta);

  [X,Y,Z]=pol2cart(Theta,R,Z);
```

```
 mesh(X,Y,Z), axis([-max(r),max(r),-max(r),max(r),-1.5,1.5])
 %pcolor(X,Y,Z), shading interp, view(2), axis('square')
 colormap('hot')
 caxis([-1,1])
 axis('off')
 title(['(',num2str(n(s)),',',num2str(m(s)),') mode'])
end
```

G. ALGORITHM c04s04a.m

```
% c04s04a.m  plot of Bessel functions
% of the 1st and 2nd
clear, figure(1), clf
order=0:3;
x=linspace(0,20,1000);
J=besselj(order,x);           % Bessel 1st kind
Y=bessely(order,x);           % Bessel 2nd kind: Neuman
subplot(211), plot(x,J)
axis([0,max(x),-1.2,1.2])
title('Bessel 1st kind (J)')
subplot(212), plot(x,Y)
axis([0,max(x),-1.2,1.2])
title('Bessel 2nd kind (Y): Neuman')
```

```
set([subplot(211),subplot(212)],'xtick',...
 [0,10,20],'ytick',[-1,0,1])
```

H. ALGORITHM c04s04b.m

```
% c04s04b.m  plots 1st thru 3rd order of Bessel function of 1st kind
% uses function bsl.m to compute zero crossing
clear, figure(1), clf
zc=5;                          % zero crossings
[J0Z,J0E]=bsl(0,zc);
[J1Z,J1E]=bsl(1,zc);
[J2Z,J2E]=bsl(2,zc);
x=linspace(0,20,200);
for n=1:3;                     % Bessel lines
  subplot(3,1,n), plot(x,besselj(n-1,x))
  line([0,max(x)],[0,0],'linestyle',':','color',[1 1 1])
end
for n=1:zc;                    % zero crossings
  subplot(311), text(J0Z(n),.1,num2str(J0Z(n)),'rot',45,'fontsize',8)
  ylabel('J0')
  subplot(312), text(J1Z(n),.1,num2str(J1Z(n)),'rot',45,'fontsize',8)
  ylabel('J1')
  subplot(313), text(J2Z(n),.1,num2str(J2Z(n)),'rot',45,'fontsize',8)
```

```
 ylabel('J2')
end
set([subplot(311),subplot(312),subplot(313)],'xtick',[0,max(x)],...
 'ytick',[-1,0,1],'xlim',[0,max(x)],'ylim',[-1.2,1.2],'fontsize',10)
```

I. ALGORITHM bsl.m

```
function [Z,E]=bsl(m,n)
%function [Z,E]=bsl(m,n)
% determines 4 element sets of upper inflection point, upper to lower
% zero crossing, lower to upper zero crossing, and lower inflection point for
% Bessel functions of the first kind any order
% m = Bessel order
% n = number of off axis null pairs and extrema pairs
% E = extrema (both upper and lower inflection points)
% Z = zeros crossing (both upper to lower and lower to upper)
accuracy=1:2; % 1=.1, 2=.0001, 3=10^-7, 4= 10^-10
              % note: every 6 decimal places doubles processing time
              % 20 extrema at 4 decimal accuracy = 12 sec on Pentium 90 Mhz
range=100;    % range within to search for 1st upper inflection
              % for every multiple of 100 greater than 10
              % accuracy decreased by 1 decimal place
lowlim=0; uplim=range;
```

```
U=[]; L=[]; Zul=[]; Zlu=[]; E=[]; Z=[];

for sets=1:ceil(n/2);                         % number of sets to find

 %%%%% determining upper inflection

 for acc=accuracy;

  x=linspace(lowlim,uplim,1000);

  [xV,xP]=max(besselj(m,x));                  % x value & x position

  dx=x(2)-x(1);                               % x spacing

  lowlim=max([0,x(xP)-dx/2]);  uplim=x(xP)+dx/2;

 end

 U=[U,x(xP)]; E=[E,x(xP)];

 %%%%% determining lower inflection

 lowlim=U(length(U));

 uplim=lowlim+range;

 for acc=accuracy

  x=linspace(lowlim,uplim,1000);

  [xV,xP]=min(besselj(m,x));

  dx=x(2)-x(1);

  lowlim=max([0,x(xP)-dx/2]);  uplim=x(xP)+dx/2;

 end

 L=[L,x(xP)]; E=[E,x(xP)];

 %%%%% determining zero crossing  (high to low)
 lowlim=U(length(U)); uplim=L(length(L));
 for acc=accuracy
  x=linspace(lowlim,uplim,1000);
  [xV,xP]=min(abs(besselj(m,x)));
  dx=x(2)-x(1);
  lowlim=max([0,x(xP)-dx/2]);  uplim=x(xP)+dx/2;
 end
 Zul=[Zul,x(xP)]; Z=[Z,x(xP)];
 %%%%% determining zero crossing  (low to high)
 lowlim=L(length(L)); uplim=lowlim+L(length(L))-U(length(L));
 for acc=accuracy
  x=linspace(lowlim,uplim,1000);
  [xV,xP]=min(abs(besselj(m,x)));
  dx=x(2)-x(1);
  lowlim=max([0,x(xP)-dx/2]);  uplim=x(xP)+dx/2;
 end
 Zlu=[Zlu,x(xP)]; Z=[Z,x(xP)];
 lowlim=Zlu(length(Zlu)); uplim=lowlim+range;
end
Z=Z(1:n);
E=E(1:n);
```

XXIV. FORCED VIBRATION OF A MEMBRANE

Algorithm c04s07.m plots the displacement of a driven circular membrane. It also graphs the displacement along a radius and the average displacement. This program and the following discussion illustrates concepts addressed in KFCS Section 4.7.

A. CONCEPT

A driven membrane is not restricted to the discrete frequencies of a free membrane. Any driving frequency may be employed, but when the driving frequency is equal to the normal mode frequency, the displacement of an ideal membrane goes to infinity.

In Figure 24.1, the upper four graphs show the normalized displacement of a circular membrane driven by a uniform pressure. The lower two graphs show that as the frequency is varied, the displacement at the center of the membrane can be very small, or as already

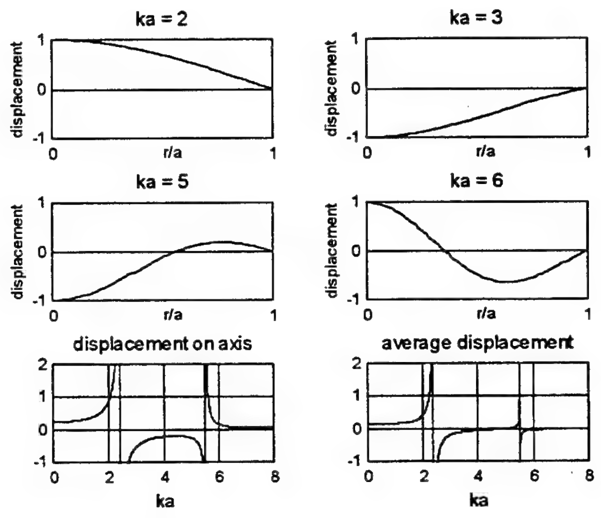

Figure 24.1 Displacement of a Driven Circular Membrane

mentioned, can go to infinity. KFCS Equation 4.46 predicts that when J_0 equals zero, the amplitude of displacement will go to infinity. The variable ka is the produce of the wave number and the radius of the circular membrane. A negative displacement denotes that the displacement is 180 degrees out of phase with the driving force. Using the function bsl.m, described in the previous chapter, the first two zero crossing were determined to occur at 2.4 and 5.2. These positions are noted on the bottom two graphs.

KFCS Equation 4.47 was used to determine the average displacement. This equation indicates that the average displacement will be zero at values of ka causing the J_2 Bessel function to cross zero. Using bsl.m it was determined that the first such crossing is at 5.1. Notice in the average displacement plot, the curve crosses zero at 5.1. Also note that while the average displacement may be zero, that does not mean that there is no displacement, as indicated by the graph of displacement against radius for ka = 5.0.

B. SALIENT FEATURES OF c04s07.m

1. Driving Frequency

The values of driving frequency for the upper four graphs are drawn from the vector "ka". Up to four values may be plotted. For example, "ka=[2.3,2.4,2.5]" will graph displacement around the first zero crossing of J_0.

2. Range of ka in Axis and Average Displacement Graphs

To use a different ka range in the lower two graphs, change the vector "ka" approximately two thirds of the way down the code. For example, "ka=linspace(0,20,500)" will include the first six J_0 crossings.

C. ALGORITHM c04s07.m

```
% c04s07.m  Forced vibration of a membrane

% note that solutions are normalized in upper four plots

clear, figure(1), clf

a=1;                                % radius at rim

r=linspace(0,a,100);                % radius increments

ka=[2,3,5,6];                       % wave number * radius

for n=1:length(ka)

 k=ka(n)/a;

 kr=k*r;

 Jkr=besselj(0,kr);

 Jka=besselj(0,ka(n));

 psi=(Jkr-Jka)/(k^2*Jka);

 subplot(3,2,n)

 plot(r,psi./max(abs(psi)))

 axis([0,1,-1,1])

 set(gca,'xtick',[0,a],'ytick',[-1,0,1],'fontsize',8)

 grid

 title(['ka = ', num2str(ka(n))],'fontsize',10)

 text(.5,-1.3,'r/a','fontsize',8)

 ylabel('displacement','fontsize',8)
```

```
end
ka=linspace(eps,8,1000);
k=ka/a;
subplot(3,2,5)
 y0=(1-besselj(0,ka))./(k.^2.*besselj(0,ka));   % axis displacement
 plot(ka,y0)
 axis([0,max(ka),-1,2])
 title('displacement on axis','fontsize',10)
 xlabel('ka','fontsize',9)
 grid
 set(gca,'xtick',0:2:10,'ytick',[-1,0,1,2],'fontsize',8)
subplot(3,2,6)
 ys=besselj(2,ka)./k.^2./besselj(0,ka);           % average displacement
 plot(ka,ys)
 axis([0,max(ka),-1,2])
 title('average displacement','fontsize',10)
 xlabel('ka','fontsize',9)
 grid
 set(gca,'xtick',0:2:10,'ytick',[-1,0,1,2],'fontsize',8)
```

XXV. VIBRATION OF THIN PLATES

Algorithm c04s08.m determines the allowable values of Ka and plots the normalized transverse displacement of a thin plate. Bessel functions and modified Bessel functions of the first kind are plotted by c04s08a.m. This program and the following discussion illustrate concepts addressed in KFCS Section 4.7.

A. CONCEPT

The vibration of a thin plate is analogous to that of a membrane, the primary difference being the nature of the restoring force.

The solution of equation of motion for a circular plate, KFCS Equation 4.55, includes both J_0 and I_0. J_0, we are already familiar with; it denotes the Bessel function of the first kind. I_0, however, represents a Bessel function, modified by multiplying the argument by the square root of negative one, making it an complex number.

This modified Bessel function is not a solution to Bessel's equation. The most striking difference between J_m and I_m can be seen in Figure 25.1, which was produced by c04s08a.m. While J_m alternates about zero, I_m is hyperbolic, it increases continuously with its argument.

Examination of KFCS Equation 4.59 shows that, for a free plate, only a discrete set of frequencies are allowed. Algorithm c04s08.m determines these frequencies, and plots the displacement for a radially symmetrical normal mode of a circular plate, as shown in Figure 25.2.

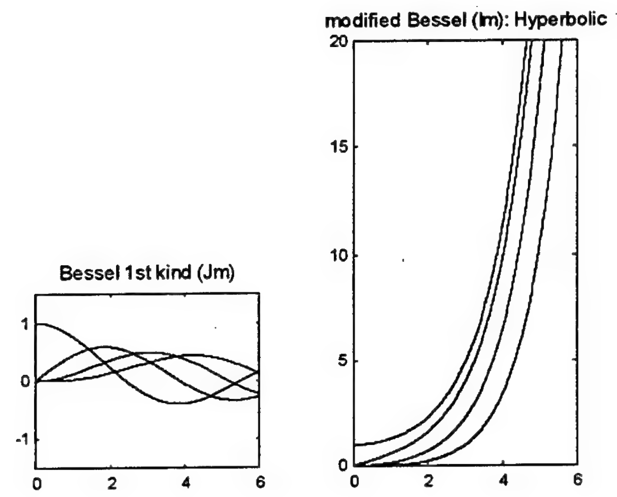

Figure 25.1 Comparison of Bessel and Modified

Bessel of the First Kind

Comparison of the normalized displacement of a circular plate with that of a circular membrane (produced by c04s07.m in the previous chapter) shows that the displacement of

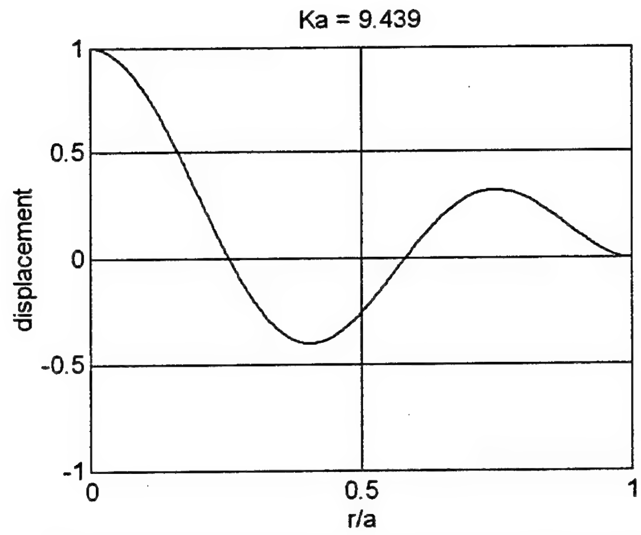

Figure 25.2 Displacement of a Driven Circular

Membrane

the plate near the edge is much smaller that of the membrane. Overall, the ratio of average amplitude to amplitude at the center is much smaller for the disk than the membrane.

B. SALIENT FEATURES OF c04s08.m

1. Allowable Values of Ka

To find the fourth allowable Ka, for example, use "n=4".

2. Accuracy

For speed of computation, the accuracy of the values of Ka determined by this algorithm has been set to 10^{-4}. To increase the accuracy, simply increase the length of vector "acc". For example, to set the accuracy to 10^{-6}, use "acc=1:6". The vector must include all integer values from one to the desired accuracy digit.

C. SALIENT FEATURES OF c04s08a.m

This algorithm is a modified version of c04s04a.m. Similar considerations apply.

D. ALGORITHM c04s08.m

```
% c04s08.m  vibration of thin plates
% determines allowable Ka and plots normalized transverse displacement
clear, figure(1), clf
n=3;                                % allowable Ka
xx=n*pi;
for acc=1:4;                        % 1e-4 accuracy
  x=linspace(xx-10^(-acc),xx+10^(-acc),200);
  [val,pos]=min(abs(besselj(0,x).*besseli(1,x)+...
   besselj(1,x).*besseli(0,x)));
  xx=x(pos);
end
Ka=x(pos);
r=linspace(0,1,1000);                              % radius
B1=-besselj(0,Ka)/besseli(0,Ka);
y=besselj(0,Ka*r)+B1*besseli(0,Ka*r);
plot(r,y)
axis([0,1,-1,1])
set(gca,'xtick',[0,.5,1],'ytick',[-1,-.5,0,.5,1])
grid
title(['Ka = ',num2str(Ka)])
xlabel('r/a'), ylabel('displacement')
```

E. ALGORITHM c04s08a.m

```
% c04s08a.m  plot of Bessel functions
% of the 1st and 2nd
clear, figure(1), clf
order=0:3;
x=linspace(0,6,1000);                              % argument
J=besselj(order,x);                                % Bessel 1st kind
I=besseli(0:3,x);                                  % modified Bessel: hyperbolic
```

```
subplot(223)
 plot(x,J)
 axis([0,max(x),-1.5,1.5])
 title('Bessel 1st kind (Jm)','fontsize',10)
 set(gca,'xtick',0:2:6,'ytick',-1:1,'fontsize',8)
subplot(122), plot(x,I)
 axis([0,max(x),0,20])
 title('modified Bessel (Im): Hyperbolic','fontsize',10)
 set(gca,'xtick',0:2:6,'ytick',0:5:20,'fontsize',8)
```

XXVI. OBLIQUE PLANE WAVE

Algorithm c05s07.m plots a three dimensional representation of a plane wave and it propagation vector. This program and the following discussion illustrate concepts addressed in KFCS Section 5.7.

A. CONCEPT

Figure 26.1 represents three surfaces of constant phase of a plane wave traveling in the direction indicated by the arrow. The direction of propagation, with respect to the origin, is determined by the x, y and z components of the wave number, k. In this case, $k_x = 1$, $k_y = -2$ and $k_z = 0$.

Since this plot is not an animation its motion is not apparent, but it's important to realize that this drawing does not portray a standing wave. The constant phase surfaces are

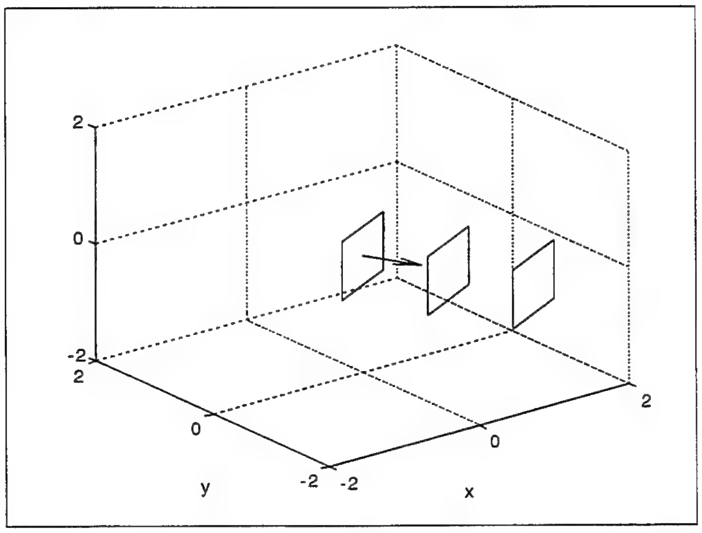

Figure 26.1 Oblique Plane Wave: k_x=-1, k_y=-2, k_z=0

analogous to the crests of a transverse wave traveling along a string aligned with propagation vector, k.

Secondly, its important to notice that plane waves do not diverge as they propagate. This is a distinctive characteristic of plane waves and provides an excellent reference by which to compare other waveforms, such as spherical or cylindrical waves.

B. ALGORITHM LIMITATIONS

The arrow in this figure represents the propagation vector, k. As discussed above, its magnitude is determined by k_x, k_y and k_z. In this algorithm, however, its magnitude is set to a constant to prevent the arrow representing large wave numbers from being drawn outside the axis boundaries of the plot.

C. SALIENT FEATURES OF c05s07.m

This algorithm is a function. As such, there is no need to change its coding to modify the wave's direction of propagation. The wave number components are included when calling the function in the MATLAB command window. To show propagation sixty degrees from the y-axis in the y-z plane, for example, type "c05s07(0,1,2)". To draw the propagation straight up, type "c05s07(0,0,1)" or "c05s07(0,0,2)", etc.. The important thing is the direction of the wave number, not its magnitude.

If no wave number components are included, such as by typing only "c05s07", the algorithm will draw the wave propagating along the positive y-axis.

D. ALGORITHM c05s07.m

```
function c05s07(kx,ky,kz)

% function c05s07(kx,ky,kz)

% plots oblique plane wave and propagation vector k

% for arbitrary wave number components, kx, ky & kz

(1), clf

if nargin<1, kx=0;ky=0;kz=0; end

k=sqrt(kx^2+ky^2+kz^2);

az=atan2(kx,ky)*180/pi;

el=asin(kz/(k+eps))*180/pi;

x=[.5,-.5,-.5,.5,.5];

y=[0,0,0,0,0];

z=[.5,.5,-.5,-.5,.5];

ln(1)=line([0,0],[0,.7],[0,0],'color',[0 1 0]);          % k line

ln(2)=line([0,.1],[.7,.4],[0,0],'color',[0 1 0]);        % arrow head 1

ln(3)=line([0,-.1],[.7,.4],[0,0],'color',[0 1 0]);       % arrow head 2

ln(4)=line(x,y,z);                                       % square 1

ln(5)=line(x,y+1,z);                                     % square 2

ln(6)=line(x,y+2,z);                                     % square 3

if k>0|k<0;

for n=1:6
```

```
 rotate(ln(n),[0 0 1],az)
end
for n=1:6
 if kx==0&ky==0; rax=[-1 0 0];
 else, rax=cross([0 0 1],[kx ky 0]);
 end
 rotate(ln(n),rax,el)
end, end
xlabel('x'), ylabel('y'), zlabel=('z');
ax=2; axis([-ax,ax,-ax,ax,-ax,ax])
set(gca,'xtick',[-ax,0,ax],'ytick',[-ax,0,ax],'ztick',[-ax,0,ax])
grid
```

XXVII. SPHERICAL WAVES

Algorithm c05s11.m compares the magnitude and phase of the specific acoustic impedance of a spherical wave to those of cylindrical and plane waves. Algorithm c05s11a.m plots the amplitude of the particle speed of a spherical wave as a function of range for various wave number values. These programs and the following discussion illustrate concepts addressed in KFCS Section 5.10 and 5.11.

A. CONCEPT

The specific acoustic impedance of a wave is composed of two parts: one real and resistive, the other imaginary and reactive.

A progressive plane wave, exhibits only resistance. A cylindrical wave diverges in two dimensions, while a spherical wave diverges in three. Notice the effect this has on their specific acoustic impedances in Figure 27.1.

The specific acoustic impedance of a traveling plane wave is a constant. It is equal to the characteristic impedance of the medium, which in this case, was chosen to be that of air. Regardless of the characteristics of the fluid media, these curves will maintain the same relationship.

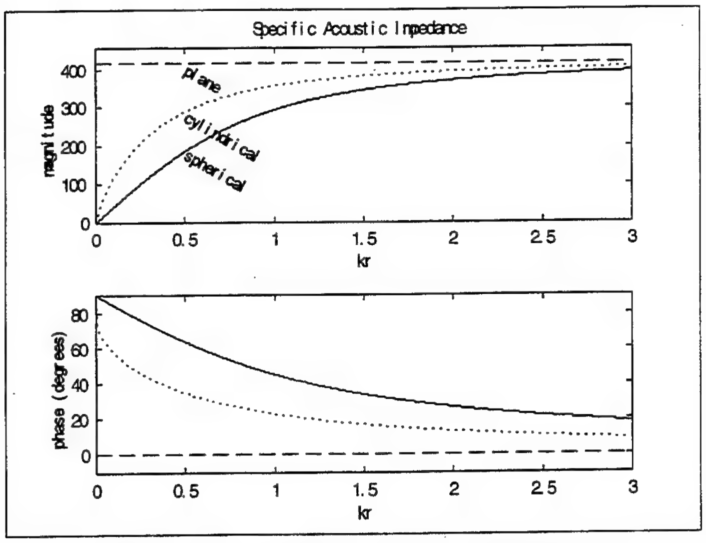

Figure 27.1 Specific Acoustic Impedance

The ordinate of these graphs is the variable, kr, which is the wave number multiplied by the range from the source. This product is proportional to the ratio of range to wavelength. For small values of kr, such as at ranges small compared to a wavelength, the magnitude of the impedance of a spherical or cylindrical wave is ver small. As shown in Figure 27.2, for a fixed pressure amplitude, small impedances result as the origin (x=0) is approached. If the range is to be kept constant, decreasing the driving frequency decreases kr, thereby increasing the particle speed (for fixed pressure amplitude).

B. LIMITING BEHAVIOR

As range from the source increases, the curvature of both spherical and cylindrical waves becomes very large and the wave can be approximated as a progressive plane waves. Accordingly, as range approaches infinity, the specific acoustic impedance should approach that of a progressive plane wave. This can be observed by plotting the graphs of c05s11.m at large values of kr.

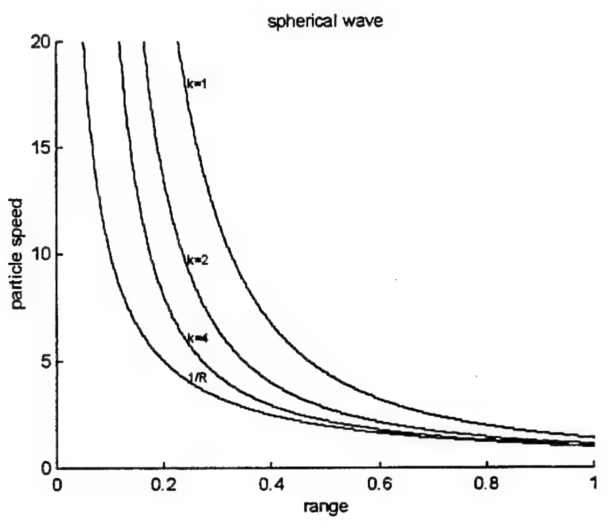

Figure 27.2 Particle Speed of Spherical Waves

In this limit, a spherical wave can approximate a plane wave, locally; the big picture must be kept in mind. A spherical wave still diverges in three dimensions. Consequently, the particle speed of a spherical wave differs from a plane in that it is inversely proportional to range from the source. Since phase goes to zero for large kr, as indicated in the Figure 27.1, large values of kr cause the pressure and particle speed to be in phase just as for a plane wave. Therefore, at large wave numbers, the particle speed is inversely proportional to the range. This can be checked in algorithm c05s11a.m by either increasing the range or the wave number. The particle velocity curves should approach the inverse range curve, labeled "1/R" in the second figure.

C. SALIENT FEATURES OF c05s11.m

1. Range to Wavelength Ratio

The ratio of range to wavelength is contained in the variable "kr". This is a unitless vector. To plot the specific acoustic impedance for very large ratios of range to wavelength, simply increase the maximum value of kr. For example, "kr=linspace(0,1000,10)", will plot out to range of a thousand times the wavelength. This range gives a good picture of spherical and cylindrical wave behavior at large ranges.

2. Characteristic Impedance

The characteristic impedance of the media is contained in the variable, "rhoc". This variable can be set to any value, but it only effects the scaling of the results.

D. SALIENT FEATURES OF c05s11a.m

1. Characteristic Impedance

As in the algorithm described above, the characteristic impedance of media only has the effect of scaling the plotted results. It this case it is set to unity, but it can be similarly changed to any value.

2. Range

The range is contained in the vector, "r". It can be increased in the same manner as the kr vector described for algorithm c05s11.m.

E. ALGORITHM c05s11.m

```
% c05s11.m  comparison specific acoustic impedance
% of spherical, cylindrical and plane waves
clear, figure(1), clf
kr=linspace(eps,3,500);           % wave number * range
rhoc=415;                         % characteristic impedance of air
%%%%%%% plane wave
```

```
Zp=rhoc*ones(size(kr));

%%%%%%% spherical wave

Rs=rhoc*kr.^2./(1+kr.^2);                % specific acoustic resistance

Xs=rhoc*kr./(1+kr.^2);                   % specific acoustic reactance

Zs=Rs+j*Xs;                              % specific acoustic impedance

%%%%%%% cylindrical wave

H0=besselj(0,kr)-j*bessely(0,kr);

H1=besselj(1,kr)-j*bessely(1,kr);

Zc=j*rhoc*H0./H1;

subplot(211)

 plot(kr,abs(Zp),'- -',kr,abs(Zs),kr,abs(Zc),':')

 xlabel('kr'), ylabel('magnitude')

 title('Specific Acoustic Impedance')

 axis([0,max(kr),0,rhoc+rhoc*.1])

 tpos=min(find(kr>.5));                  % text position

 text(kr(tpos),abs(Zp(tpos)-rhoc*.02),'plane','rot',-30)

 text(kr(tpos),abs(Zc(tpos)-rhoc*.02),'cylindrical','rot',-30)

 text(kr(tpos),abs(Zs(tpos)-rhoc*.02),'spherical','rot',-30)

subplot(212)

 plot(kr,angle(Zp)*180/pi,'- -',kr,angle(Zs)*180/pi,...

  kr,angle(Zc)*180/pi,':')
 xlabel('kr'), ylabel('phase (degrees)')
 axis([0,max(kr),-10,90])
```

F. ALGORITHM c05s11a.m

```
% c05s11a.m  particle speed of a spherical wave
% as a function of range for various wave numbers
clear, figure(1), clf
rhoc=1;
r=linspace(eps,1,1000);                 % range
for k=[1,2,4];                          % wave number
  theta=acot(k*r);                      % phase angle
  U=1./r./cos(theta);                   % speed amplitude
  line(r,U,'linestyle',':'),            % speed line
  text(r(240)+.005,U(240),['k=',num2str(k)],'fontsize',9)
end
line(r,1./r),                           % 1/r line
text(r(240)+.005,1./r(240),'1/R','fontsize',9)
axis([0,max(r),0,20])
set(gca,'ytick',0:5:20)
title('spherical wave')
xlabel('range'), ylabel('particle speed')
```

XXVIII. TRANSMISSION FROM ONE FLUID TO ANOTHER: NORMAL INCIDENCE

Algorithm c06s02.m plots the reflection and transmission coefficients for both pressure and intensity of a wave traveling from one fluid to another at normal incidence. This program and the following discussion illustrate concepts addressed in KFCS Section 6.2.

A. CONCEPT

Pressure, intensity and power transmission and reflection coefficients are defined as the ratio of the pressure, intensity or power of the transmitted or reflected wave to that of the incident wave. These coefficients are plotted in Figure 28.1 as a function of the ratio of characteristic acoustic impedances of two fluid media, r1 and r2. Since for normal incidence,

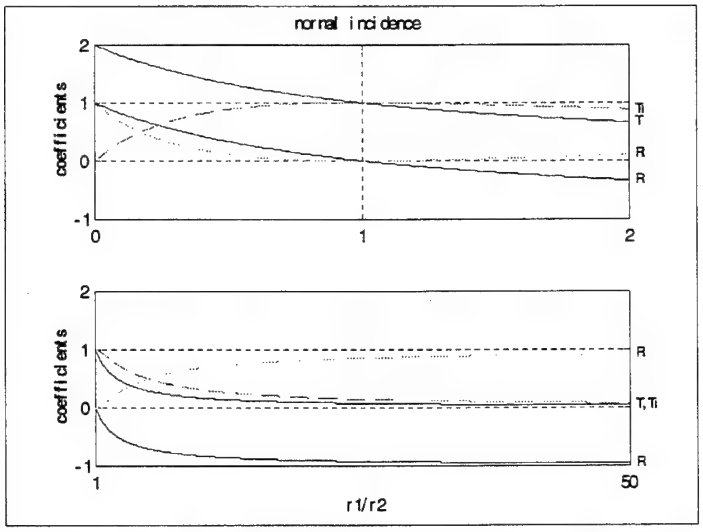

Figure 28.1 Fluid to Fluid Transmission at Normal Incidence

shown. The two graphs are the same, except they show the different range of the ratio of characteristic impedances.

The pressure reflection coefficient, R, is always real. Notice that when r1 < r2, R is positive, but when r1 > r2, R is negative. This negative value represents a 180° phase change of the reflected wave.

Since the pressure transmission coefficient, T, is real and positive, the transmitted wave is always in phase with the incident wave. The relationship between the pressure coefficients is T = R + 1. Therefore, as shown in the graphs, the maximum amplitude of the transmission coefficient is two.

Ti and Ri range from zero to unity, but since the areas of the beams are equal and power is conserved, they sum to unity; Ti + Ri = 1.

B. LIMITING BEHAVIOR

1. Rigid Boundary

At a perfectly rigid boundary, such as a sound wave in air striking a hard wall, the characteristic impedance of the second media is much larger than the first. This boundary condition is such that the particle velocity goes to zero at the the boundary. Since the reflected pressure is in phase with the incident wave, these pressures combine to produce in the wall, a pressure amplitude that is twice the maximum amplitude of the incident wave alone. However, energy doesn't propagate into the wall, since the particle speed is zero and the intensity is, therefore, zero. These coefficients can be observed on the upper plot at r1/r2 = 0.

2. Equal Characteristic Impedances

When r1 = r2, it's as if there is no acoustic boundary. The wave should be completely transmitted with no reflection. This can be observed in the upper plot at r1/r2 = 1.

3. Free (Pressure Release) Boundary

At a free boundary, such as a sound wave in water, normally incident to the surface of the ocean, all of the wave is reflected (R = Ri = 1) and none transmitted (T = Ti = 0). This can be observed at large characteristic impedance ratios, such as in the lower plot at r1/r2 = 50. Notice also the 180° phase shift of the reflected pressure. This destructive combination results in zero pressure amplitude at the boundary. This type of boundary is frequently call a pressure release boundary.

C. ALGORITHM c06s02.m

```
% c06s02.m  reflection at normal incidence
% plots transmission and reflection coefficients
% for both pressure and intensity
clear, figure(1), clf
subplot(211)
r=linspace(eps,2,200);          % char impedance ratio r1/r2
R=(1-r)./(1+r);                 % pressure reflection coeff
T=R+1;                          % pressure transmission coeff
Ri=R.^2;                        % intensity refllection coeff
Ti=1-Ri;                        % intensity transmission coeff
```

```
plot(r,R,r,T,r,Ri,'- -',r,Ti,'- -'), grid

[m,l]=max(r);

text([m,m,m,m],[Ti(l),Ri(l),T(l),R(l)],[' Ti';' Ri';' T ';' R '],...

 'fontsize',10)

set(gca,'ytick',[-1:2],'xtick',[0,1,max(r)])

axis([0,max(r),-1,2])

title('normal incidence')

ylabel('coefficients')

subplot(212)

r=linspace(1,50,1000);

R=(1-r)./(1+r);

T=R+1;

Ri=R.^2;

Ti=1-Ri;

plot(r,R,r,T,r,Ri,'- -',r,Ti,'- -'), grid

[m,l]=max(r);

text([m,m,m],[Ri(l),T(l),R(l)],[' Ri  ';' T,Ti';' R   '],...

 'fontsize',10)

set(gca,'ytick',[-1:2],'xtick',[1,max(r)])

axis([min(r),max(r),-1,2])

xlabel('r1/r2')
```

```
ylabel('coefficients')
```

XXIX. TRANSMISSION THROUGH A LAYER: NORMAL INCIDENCE

Algorithm c06s03.m plots the intensity transmission coefficients of a wave at normal incidence to a fluid layer. This program and the following discussion illustrate concepts addressed in KFCS Section 6.3.

A. CONCEPT

Consider the geometry involved in a wave obliquely crossing two boundaries. As the incident wave reaches the first boundary, part is reflected and part of it is transmitted to the second boundary. This transmitted part is then reflected and transmitted into the third media. The reflected part of the wave which entered the third media is returned to the first boundary from the opposite direction where it is in turn reflected and transmitted. This reflected part is returned to the second boundary where it is also partially transmitted into the third media. The wave which eventually passes into the third media is the sum of an infinite number of partial transmissions and reflections.

Figures 29.1 and 29.2 show four arrangements of dissimilar fluids. In each case, a plane wave in medium 1 is normally incident on a layer, medium 2, and exits into a third fluid, medium 3. The boundaries separating the fluids are parallel, so we need not consider oblique incidence at this point. The media are described by the ratio of their specific acoustic impedances: r12 = r1/r2 and r13 = r1/r3.

The four arrangements were chosen with particular care. You will notice that two

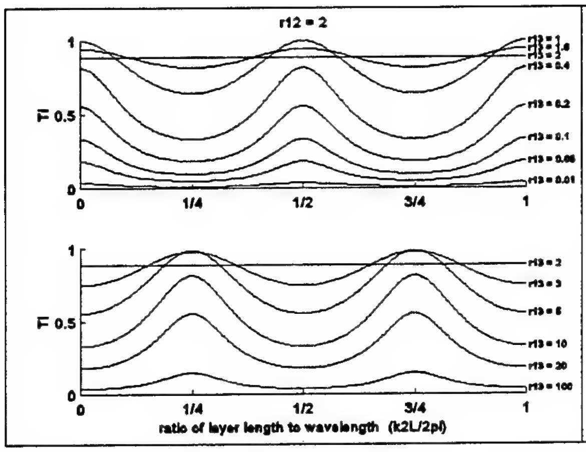

Figure 29.1 Intensity Transmission

Coefficients for r12 = 2

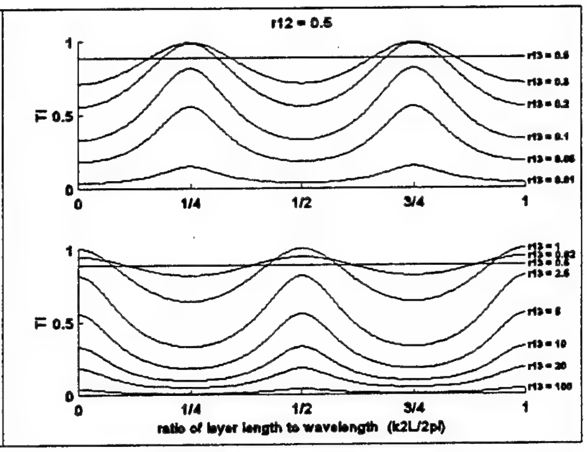

Figure 29.2 Intensity Transmission

Coefficients for r12 = 0.5

pairs of graphs show identical transmission coefficient curves. For example, the upper graph of r12 = 0.5 and the lower graph of r12 = 2 have r12 ratios are reciprocals of each other. Additionally, all r13 ratios of the former graph are reciprocals of the latter. By arranging the characteristic impedances as reciprocals of each other, we have flipped the medium around. The first arrangement would be that seen by a sound wave traveling from medium 1 to medium 3. The reciprocal arrangement would be that seen by a wave traveling in the reverse direction. The fact that both directions of travel present identical transmission coefficients is significant in that it demonstrates a special case of the principle of acoustic reciprocity.

Taking these similarities into account, there are basically two types of arrangements: one in which the impedance of the layer is between that of the surrounding media (r1 > r2 > r3 and r1 < r2 < r3) and the other in which the layer is not (r1 > r2 <r3) and r1 < r2 > r3). The remainder of this chapter will consider only the former, more common case.

If the layer has an impedance between that medium 1 and 3, it is most transparent to sound waves when the layer thickness is an odd multiple quarter wavelength of the sound in layer 2. It's most opaque when its thickness is in multiples of a half wavelength. The thickness and impedance of a layer is important when designing systems that must be separated from their medium. For example, the sonar system of a ship is separated from the surrounding ocean by a rubberized dome. If the sonar were to employ high enough frequencies, the thickness and impedance of the dome could significantly reduce sound transmission.

Note that the ratio of layer length to wavelength is the same as $k_2L/2\pi$. In other words, the medium is most transparent when k_2L is in multiples of $\pi/2$.

B. LIMITING BEHAVIOR

1. Removing the Layer

If the layer were removed, so the ratio of thickness to wavelength is zero (at the abscissa), the transmission coefficients are determined, as in the previous chapter, by the ratio of r13 alone. This can also be observed by setting r12 = 1. The coefficients become constant.

C. SALIENT FEATURES OF c06s03.m

1. Ratio of Layer Length to Wavelength

As it is currently set, the range of the r12 ratio is from zero to unity. If the range is extended, by changing the vector "k2L", it will be observed that the coefficient curves are periodic and simply repeat themselves in multiples of one half the ratio of layer length to wavelength.

2. Characteristic Acoustic Impedances of the Medium

Each time this algorithm is run, it produces two graphs. The values of r13 for the upper graph are contained in the vector "r13U". The values for the lower graph are in the vector "r13L". To modify the r12 ratio, change the variable, "r12".

The r13U, r13L, and r12 values used for both figures shown are included in the code. One set is simply negated by the "%" symbol preceding the line of code.

D. ALGORITHM c06s03.m

```
% c06s03.m transmission through a layer at normal incidence
% plots intensity transmission coefficients as a function of
% layer length to wavelength (k2l/2pi) for various
% characteristic acoustic impedances, r1, r2, and r3
clear, figure(1), clf
r12=2;                                      % r1 > r2 case 1
r13U=[.01,.05,.1,.2,.4,1,1.6,2];            % r1 < r3 case 1
r13L=[2,3,5,10,20,100];                     % r1 > r3 case 1
%r12=.5;                                    % r1 < r2 case 2
%r13U=[.01,.05,.1,.2,.3,.5];                % r1 < r3 case 2
%r13L=[.5,.62,1,2.5,5,10,20,100];           % r1 > r3 case 2
k2L=linspace(0,2*pi,500);                   % kL of layer
for n=1:2
 subplot(2,1,n)
```

```
if n==1, r13=r13U; elseif n==2, r13=r13L; end
for m=1:length(r13)
 r23=r13(m)/r12;                                    % r2/r3
 Rn=(1-r13(m)).*cos(k2L)+j*(r23-r12).*sin(k2L);  % numerator
 Rd=(1+r13(m)).*cos(k2L)+j*(r23+r12).*sin(k2L); % denominator
 R=Rn./Rd;                                          % intensity reflection coefficient
 Ti=4./(2+((1/r13(m))+r13(m)).*cos(k2L).^2+...
  ((1/r12)*r23+r12*(1/r23)).*sin(k2L).^2);         % intensity transmission coeff
 line(k2L/2/pi,Ti)
 text(max(k2L)/2/pi,Ti(length(Ti)),[' r13 = ',num2str(r13(m))],'fontsize',9)
 end
 axis([0,max(k2L)/2/pi,0,1])
 set(gca,'xtick',[0,1/4,1/2,3/4,1,2,3],'xticklabels',...
  [' 0 ';'1/4';'1/2';'3/4';' 1 ';' 2 ';' 3 '])
 ylabel('Ti')
end
subplot(211), title(['r12 = ',num2str(r12)])
subplot(212), xlabel('ratio of layer length to wavelength  (k2L/2pi)')
```

XXX. TRANSMISSION FROM ONE FLUID TO ANOTHER: OBLIQUE INCIDENCE

Algorithm c06s04.m produces an animation which graphically demonstrates the behavior of a wave incident on a boundary at oblique incidence. The resulting pressure reflection coefficient is plotted as a function of angle of incidence in algorithm c06s04a.m. These programs and the following discussion illustrate concepts addressed in KFCS Section 6.4.

A. CONCEPT

A pressure wave striking the boundary between two dissimilar fluids is reflected back into the first medium and transmitted into the second medium. This is demonstrated in the Figure 30.1, which is one frame of an animation produced by c06s04.m. The solid vertical line represents the boundary between two fluids of dissimilar sound speeds, labeled c1 and c2.

The horizontal, dotted line, is a reference line drawn normal to the fluid boundary. In KFCS, the angles of incidence, reflection and transmission are expressed with respect to this normal reference line. In some acoustics texts, the convention is to express these angles as grazing angles. Care must be taken to identify the convention in use because it alters the form of Snell's law.

The oblique, dotted line, when present, represents the critical angle. The critical angle

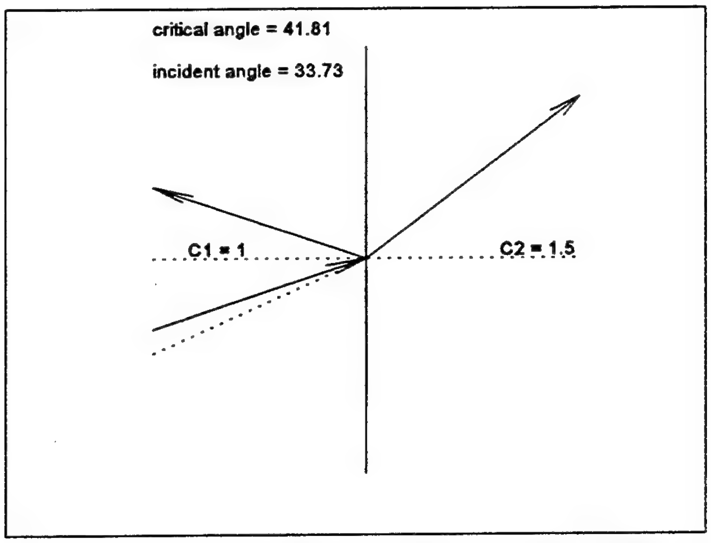

Figure 30.1 Fluid to Fluid Transmission with Oblique Incidence

is the angle beyond which the wave is not transmitted into the second medium. It is a function of the ratio of media sound speeds. The critical angle only exists when c1<c2.

Algorithm c06s04.m animates the incident, transmitted, and reflected wave directions by incrementing the angle of incidence from zero to 90 degrees. When c1 < c2 and the angle of incidence is less than the critical angle, the transmitted wave bends away from the normal. When incidence is greater than critical, the transmitted wave propagates parallel to the boundary; the incident wave is totally reflected. When c1 > c2, the transmitted wave bends towards normal.

Figure 30.2 was produced by c06s04a.m. The two left hand graphs are for a wave approaching a slower medium (c1 > c2). The pressure reflection coefficient is always real, and the angle of transmission is less than the angle of incidence. In other words, the transmitted wave bends towards the normal, as illustrated by c06s04.m.

Notice that when the critical angle of incidence is reached, the reflection coefficient maintains a constant value of unity. All of the incident wave is reflected.

An important thing to notice in the bottom two graphs of Figure 30.2 is the angle at which the reflection coefficient touches zero. This angle is termed the angle of intromission. Angles of intromission exist only when r1 < r2 and c1 >c2 or when r1 > r2 and c1 < c2.

B. LIMITING BEHAVIOR

When the acoustic characteristics of medium 1 is the same as medium 2, there is no reflection; the magnitude of the reflection coefficient is zero. This can be observed by setting c1 = c2 in algorithm c06s04.m and by setting c1 = c2 and r1 = r2 in algorithm c06s04a.m

When the angle is incidence is zero, the wave is normally incident and these algorithms reduce to those discussed in the previous chapter. display the characteristics

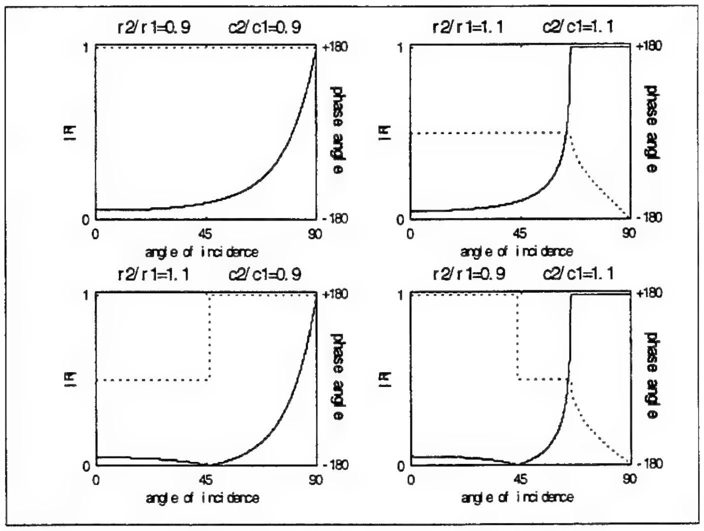

Figure 30-2 Pressure reflection coefficients for oblique incidence

reduce to those discussed in the previous chapter. display the characteristics discussed in the previous two chapters. In the case of c06s04.m, the directions of the incident, reflected, and transmitted waves will line up normal to the boundary. The intensity transmission coefficient curve at zero degrees of incidence is displayed in the third figure of this chapter. It is identical to that produced by c06s03.m for normal incidence.

C. ALGORITHM LIMITATIONS

Algorithm c06s04.m will not produce an accurate vector display for angles of incidence greater than 90 degrees. In other words, the program will not draw a vector originating in medium two.

D. SALIENT FEATURES OF c06s04.m

The sound speed of the two media is contained in the variables c1 and c2.

E. SALIENT FEATURES OF c06s04a.m

1. Characteristic Acoustic Impedance Ratio

The ratio r2/r1 is contained in the vector, "r". This vector must contain four elements, each of which is used in one of the four subplots.

2. Sound Speed Ratio

As with the impedance ratio, the vector "c" contains the c2/c1 ratios and must contain four elements.

F. ALGORITHM c06s04.m

```
% c06s04.m  fluid to fluid transmission with oblique incidence

clear, figure(1), clf, set(gcf,'backingstore','off')
```

```
th1=0;                                   % incidence angle (from normal)
c1=1;                                    % sound speed media 1
c2=1.5;                                  % sound speed media 2
thr=pi-th1;                              % reflection angle
tht=real(asin(c2/c1*sin(th1)));          % transmitted angle
if c1<c2;                                % critical angle
 thc=num2str(asin(c1/c2)*180/pi);
 line([0,1],[-tan(asin(c1/c2)),0],...
  'erasemode','background','linestyle',...
  ':','color',[1 1 1]);             % critical line
else, thc='NA'; end
x0=0; xi=1;                         % incident ray x-coord
y0=-(xi-x0)*tan(th1); yi=0;         % incident ray y-coord
Li=line([x0,xi],[y0,yi],'erasemode','xor');
xt=2; yt=(xt-xi)*tan(tht);          % transmitted ray coord
Lt=line([xi,xt],[yi,yt],'erasemode','xor');
line([xi,xi],[-2,2],'color',[1 1 1],...
 'erasemode','background');         % boundary line
line([0,2],[0,0],'linestyle',':','color',...
 [1 1 1],'erasemode','background');        % normal line
axis([0,2,-2,2]), axis('square'), axis('off')
```

```
Tinc=text(0,1.8,['incident angle = ',num2str(th1*180/pi)],...
 'erasemode','xor','fontsize',10);
text(0,2.2,['critical angle = ',thc],'erasemode',...
 'background','fontsize',10);
text(1.8,.1,['C2 = ',num2str(c2)],'fontsize',10,'horiz','center');
text(.3,.1,['C1 = ',num2str(c1)],'fontsize',10,'horiz','center');
Lia1=line([xi,xi-.2*cos(th1-.2)],[yi,yi-.2*sin(th1-.2)],...
 'erasemode','xor');                          % incident arrow
Lia2=line([xi,xi-.2*cos(th1+.2)],[yi,yi-.2*sin(th1+.2)],...
 'erasemode','xor');
Lta1=line([xt,xt-.2*cos(tht-.2)],[yt,yt-.2*sin(tht-.2)],...
 'erasemode','xor');                          % tranmitted arrow
Lta2=line([xt,xt-.2*cos(tht+.2)],[yt,yt-.2*sin(tht+.2)],...
 'erasemode','xor');
if c1>c2|c1<c2;                               % tests if two different media
  xr=x0; yr=(xr-xi)*tan(thr);                 % reflected ray coord
  Lr=line([xi,xr],[yi,yr],'erasemode','xor');
  Lra1=line([xr,xr-.2*cos(thr-.2)],[yr,yr-.2*sin(thr-.2)],...
   'erasemode','xor');                        % reflected arrow
  Lra2=line([xr,xr-.2*cos(thr+.2)],[yr,yr-.2*sin(thr+.2)],...
   'erasemode','xor');
```

```
end

drawnow

%%%%%%% cycle through range of incident angles

for th1=linspace(eps,pi/2,500),                 % incident angle

 thr=pi-th1;                                    % reflection angle

 tht=real(asin(c2/c1*sin(th1)));                % transmitted angle

 y0=-(xi-x0)*tan(th1); yi=0;                    % incident ray y-coord

 set(Li,'xdata',[x0,xi],'ydata',[y0,yi]);

 yt=(xt-xi)*tan(tht);                           % transmitted ray coord

 set(Lt,'xdata',[xi,xt],'ydata',[yi,yt]);

 set(Lia1,'xdata',[xi,xi-.2*cos(th1-.2)],...

  'ydata',[yi,yi-.2*sin(th1-.2)])               % incident arrow

 set(Lia2,'xdata',[xi,xi-.2*cos(th1+.2)],...

  'ydata',[yi,yi-.2*sin(th1+.2)]);

 set(Lta1,'xdata',[xt,xt-.2*cos(tht-.2)],...

  'ydata',[yt,yt-.2*sin(tht-.2)]);              % tranmitted arrow

 set(Lta2,'xdata',[xt,xt-.2*cos(tht+.2)],...

  'ydata',[yt,yt-.2*sin(tht+.2)]);

 if c1>c2|c1<c2;                                % tests if two different media

  yr=(xr-xi)*tan(thr);                          % reflected ray coord

  set(Lr,'xdata',[xi,xr],'ydata',[yi,yr]);
  set(Lra1,'xdata',[xr,xr-.2*cos(thr-.2)],...
    'ydata',[yr,yr-.2*sin(thr-.2)]);              % reflected arrow
  set(Lra2,'xdata',[xr,xr-.2*cos(thr+.2)],...
    'ydata',[yr,yr-.2*sin(thr+.2)]);
 end
 set(Tinc,'string',['incident angle = ',num2str(th1*180/pi)])
 drawnow
end
```

G. ALGORITHM c06s04a.m

```
% c06s04a.m  fluid to fluid transmission with oblique incidence
% plots magnitude and phase of pressure reflection coefficient
% as function of incident angle
clear, figure(1), clf
r=[.9,1.1,1.1,.9];                                % r2/r1
c=[.9,1.1,.9,1.1];                                % c=c2/c1
th1=linspace(0,pi/2,1000);                        % incident angle
for n=1:4
 th2=asin(c(n).*sin(th1));                        % transmitted angle
 R=(r(n)-cos(th2)./cos(th1))./(r(n)+cos(th2)./cos(th1))+1e-10;
 subplot(2,2,n)
 plot(th1*180/pi,abs(R),th1*180/pi,angle(R)*180/pi/360+.5,'w:')
 axis([0,90,0,1.02])
 set(gca,'xtick',[0,45,90],'ytick',[0 1],'fontsize',10)
 title(['r2/r1=',num2str(r(n)),'    c2/c1=',num2str(c(n))],'fontsize',12)
 xlabel('angle of incidence','fontsize',10)
 ylabel('|R|')
 text(100,.5,'phase angle','horiz','center','rot',-90)
 text(92,0,'-180','fontsize',10)
 text(92,1,'+180','fontsize',10)
end
```

APPENDIX. ALGORITHMS CONCERNING RADIATION AND RECEPTION OF ACOUSTIC WAVES

This appendix contains algorithms which illustrate concepts addressed in KFCS Chapter 8. They are provided without supporting or interpretation of the acoustic processes involved.

A. ALGORITHM c08s01.m

This algorithm produced Figure 8.1. It graphs the characteristic acoustic impedance for a pulsating sphere, comparing the exact and long wavelength solutions.

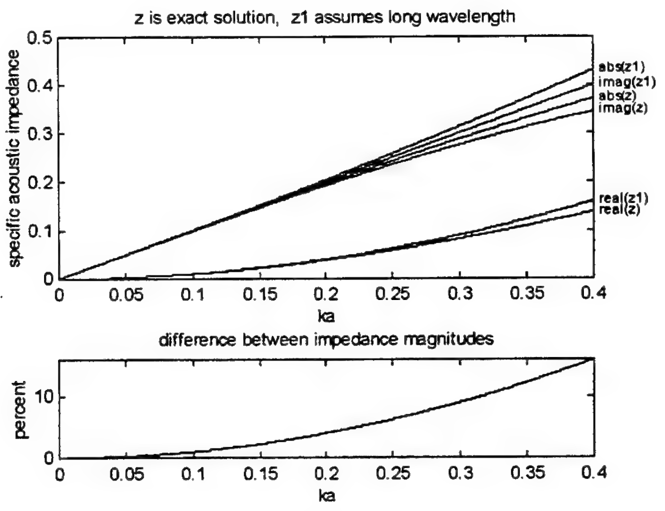

Figure 8.1 Characteristic Acoustic Impedance of a

Pulsating Sphere

```
% c08s01.m  radiation of a sphere
% methods of determining characteristic acoustic impedance
% compares exact solution with long wavelength solution
clear, figure(1), clf
```

```
ka=linspace(eps,.4,1000);

theta=acot(ka);                             % phase at r = a

z=cos(theta).*exp(j*theta);                 % impedance on surface

z1=ka.*(j+ka);                              % impedance assuming ka<<1

dz=(abs(z1)-abs(z))./abs(z)*100;            % percent difference in modulus

axes('position',[0.13 0.455 0.775 0.47])

plot(    ka,real(z),'c',ka,real(z1),':c',...

         ka,imag(z),'m',ka,imag(z1),':m',...

         ka,abs(z), 'y',ka,abs(z1), ':y')

title('z is exact solution,  z1 assumes long wavelength')

xlabel('ka'), ylabel('specific acoustic impedance')

text(max(ka)*1.01,max(abs(z)),'abs(z)','fontsize',10)

text(max(ka)*1.01,max(abs(z1)),'abs(z1)','fontsize',10)

text(max(ka)*1.01,max(imag(z)),'imag(z)','fontsize',10)

text(max(ka)*1.01,max(imag(z1)),'imag(z1)','fontsize',10)

text(max(ka)*1.01,max(real(z)),'real(z)','fontsize',10)

text(max(ka)*1.01,max(real(z1)),'real(z1)','fontsize',10)

subplot(313)

plot(ka,dz)

title('difference between impedance magnitudes')

xlabel('ka'), ylabel('percent')

axis([0,max(ka),0,ceil(max(dz))])
```

B. ALGORITHM c08s05.m

This algorithm produced Figure 8.2. It compares the exact and dipole pressure solutions of an acoustic doublet.

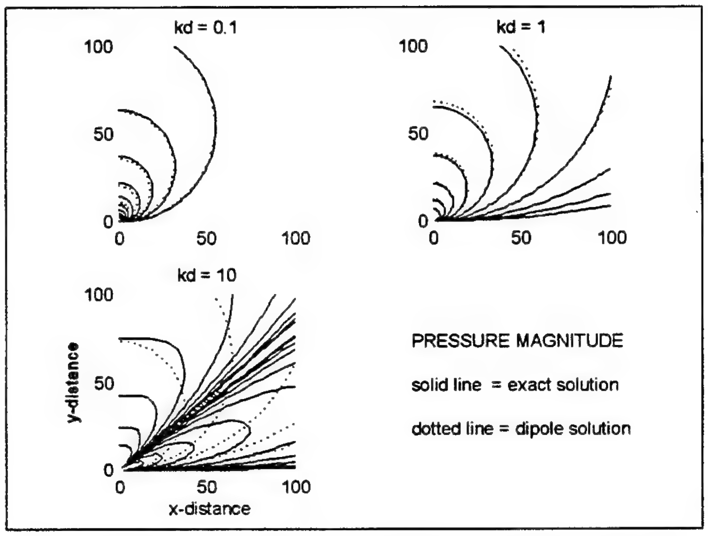

Figure 8.2 Exact and Dipole Pressure Solutions of Acoustic Dipole

```
% c08s05.m Doublet
% comparison of exact and dipole pressure solutions
% for sources vertically separated by "d" meters
clear, figure(1), clf
k=[0.1,1,10];                       % wave number
d=1;                                % source separation
y1=-d/2;                            % source 1 position
y2=d/2;                         % source 2 position
x=logspace(-1,2,100);           % x-distance from center of sources
y=logspace(-1,2,100);           % y-distance from center of sources
```

```
[X,Y]=meshgrid(x,y);
r=sqrt(X.^2+Y.^2);                              % range from midpoint
r1=sqrt(X.^2+(Y+y1).^2);                        % range from source 1
r2=sqrt(X.^2+(Y+y2).^2);                        % range from source 2
v=linspace(-7,-2,10);                           % contour lines
for  n=1:length(k)
  P=abs(exp(-j*k(n)*r1)./r1-exp(-j*k(n)*r2)./r2);     % exact
  P1=(k(n)./r).*(Y./r);                         % dipole
  subplot(2,2,n)
  % note: natural log of pressures plotted for better resolution
  contour(x,y,log(P),v), hold on
  contour(x,y,log(P1),v,':'), hold off
  title(['kd = ',num2str(k(n)*d)])
  set(gca,'xtick',[0,50,100],'ytick',[0,50,100])
  axis([min(x),max(x),min(y),max(y)])
  axis('square')
end
xlabel('x-distance'), ylabel('y-distance')
subplot(224)
text(0,3,'PRESSURE MAGNITUDE')
text(0,2,'solid line  = exact solution')
text(0,1,'dotted line = dipole solution')
```

```
axis([0,3,0,4]), axis('off')
```

C. ALGORITHM c08s05a.m

This algorithm produced Figure 8.3. It examines the beam pattern of an acoustic dipole.

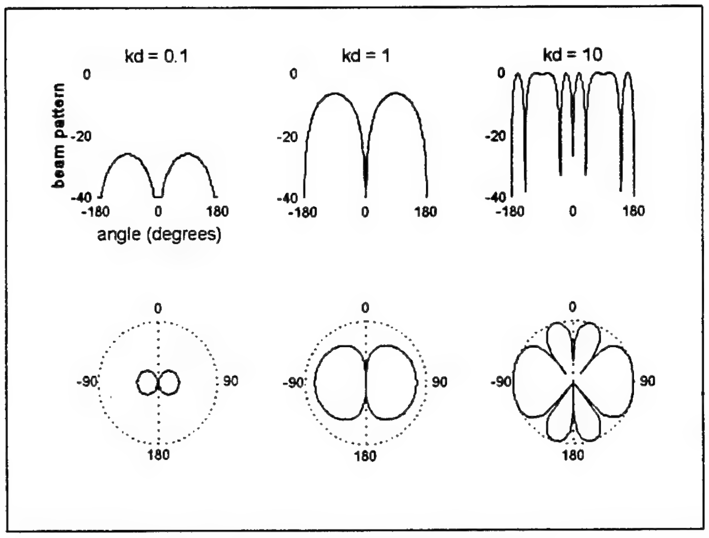

Figure 8.3 Dipole Beam Pattern

```
% c08s05a.m  dipole beam pattern
clear, figure(1), clf
kd=[0.1,1,10];
theta=linspace(-pi,pi,360);
for n=1:3;
 H=abs(sin(kd(n)/2*sin(theta)));          % directional factor
 b=20*log10(H);                           % beam pattern
 nul=find(b<-40);                         % finds values below 40 dB
 b(nul)=b(nul)-b(nul)-40;                 % sets to -40 if below 40 dB
 subplot(2,3,n)
```

```
plot(theta*180/pi,b);                              % cartesian plot
title(['kd = ',num2str(kd(n))])
axis([min(theta*180/pi),max(theta*180/pi),-40,0])
axis square
set(gca,'xtick',[-180,0,180],'fontsize',10)
subplot(2,3,n+3)
[px,py]=pol2cart(theta,b+40);                      % polar plot
plot(px,py)
% polar coordinate reference grid
text([0,48,0,-48],[48,0,-48,0],['-90';' 0 ';' 90';'180'],...
 'horiz','center','vert','middle','fontsize',10)
arcx=40*[cos(linspace(0,2*pi,200)),-1];
arcy=40*[sin(linspace(0,2*pi,200)),0];
line(arcx,arcy,'linestyle',':','color',[1 1 1])
axis equal, axis square, axis off
view([-90,90])
end
subplot(231)
xlabel('angle (degrees)','fontsize',12)
ylabel('beam pattern','fontsize',12)
```

D. ALGORITHM c08s06.m

This algorithm produced Figure 8.4. It graphs the pressure level field of a line source, comparing the exact and far field solutions.

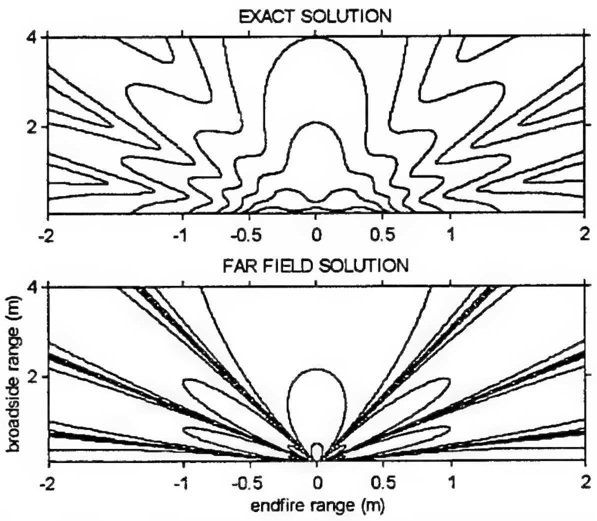

Figure 8.4 Pressure Level Field of a Line Source

```
% c08s06.m  line source: pressure level field
% determined by exact solution and far field solution
% Note: source lies at origin along x-axis
clear, figure(1), clf
k=20;                               % wave number
dx=.05;                             % differential component size
L=1;                                % line source length
x=linspace(-2,2,200);               % endfire range
y=linspace(.05,4,200);              % broadside range
[X,Y]=meshgrid(x,y);
%%%%%%% exact solution (summation of differential components)
```

```
dP=0;                                        % initializing pressure

for x1=-L/2:dx:L/2;

 r1=sqrt((X-x1).^2+(Y.^2));  % range to field point

 dP=exp(j*k*r1)./r1.*dx+dP;                  % pressure summation

end

P=20*log10(abs(dP));

nul=find(P<-40);                             % find levels below 40 dB

P(nul)=P(nul)-P(nul)-40;                     % set to -40 if below 40 dB

P=(P+40)./(max(max(P))+40);                  % make positive & normalize

%%%%%%% far field solution

th=atan2(Y,X);

r=sqrt(X.^2+Y.^2);

v=k*cos(th)/2;

H=abs(sin(v)./v);

Pax=exp(j*k*r).*L./r;

Pf=log10(abs(Pax.*H));

nul=find(Pf<-40);                            % find levels below 40 dB

Pf(nul)=Pf(nul)-Pf(nul)-40;                  % set to -40 if below 40 dB

Pf=(Pf+40)./(max(max(Pf))+40);               % make positive & normalize

%%%%%%% plotting

subplot(211)

contour(x,y,P,8)
```

```
title('EXACT SOLUTION')

subplot(212)

contour(x,y,Pf,8)

title('FAR FIELD SOLUTION')

xlabel('endfire range (m)')

ylabel('broadside range (m)')

set([subplot(211),subplot(212)],'ytick',[0,2,4],...

 'xtick',[-2,-1,-.5,0,.5,1,2],'tickdir','out')
```

E. ALGORITHM c06s06a.m

This algorithm produced Figure 8.5. If graphs the beam pattern of a line source, comparing the exact and far field solutions.

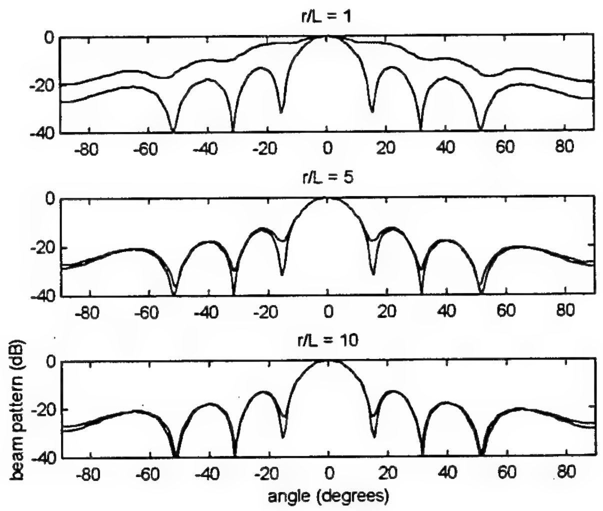

Figure 8.5 Line Source Beam Pattern: Exact and Far Field Solutions

```
% c08s06a.m  line source:  comparison of beam patterns

% determined by exact and far field methods at various ranges
% Note: source lies at origin along x-axis
clear, figure(1), clf
k=24;                                        % wave number
dx=.01;                                      % differential component size
L=1;                                         % source length
r=[1,5,10];                                  % range from source center
theta=linspace(-pi/2,pi/2,180);
for n=1:length(r);
 y=r(n)*cos(theta);
 x=r(n)*sin(theta);
 dP=0;
 for x1=-L/2:dx:L/2;
  r1=sqrt((x-x1).^2+(y.^2));
  dP=exp(j*k*r1)./r1.*dx+dP;                 % component summation
 end
 b=20*log10(abs(dP./max(dP)));               % exact solution
 nul=find(b<-40);                            % find levels below 40 dB
 b(nul)=b(nul)-b(nul)-40;                    % set to -40 if below 40 dB
 v=k*L*sin(theta)./2;
 H=abs(sin(v)./v);
 b1=20*log10(H);                             % far field solution
 nul=find(b1<-40);                           % find levels below 40 dB
 b1(nul)=b1(nul)-b1(nul)-40;                          % set to -40 if below 40 dB
 subplot(3,1,n)
 plot(theta*180/pi,b,'w:',theta*180/pi,b1,'y')
 title(['r/L = ',num2str(r(n))])
 axis([min(theta)*180/pi,max(theta)*180/pi,-40,0])
end
xlabel('angle (degrees)')
ylabel('beam pattern (dB)')
```

F. ALGORITHM c08s06b.m

This algorithm produced Figure 8.6. It examines the beam pattern of a line source.

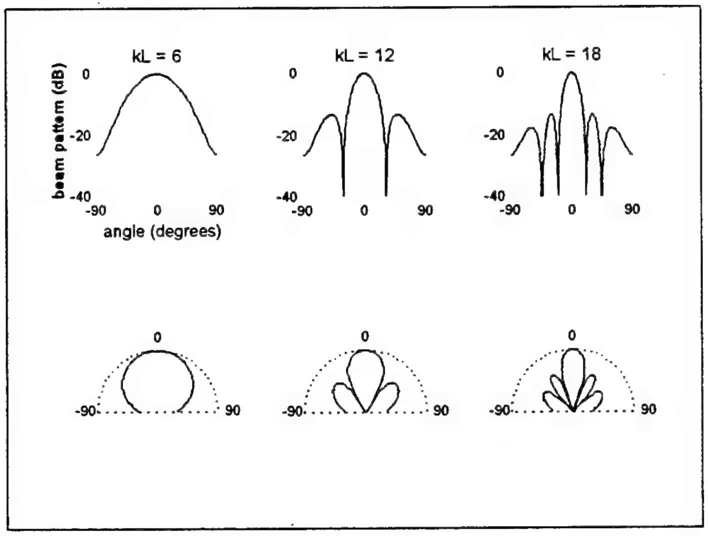

Figure 8.6 Line Source Beam Pattern

```
% c08s06b.m  line source:  beam pattern
clear, figure(1), clf
kL=[6,12,18];
theta=linspace(-pi/2,pi/2,180);
```

```
for n=1:3
 v=kL(n)*sin(theta)/2;
 H=abs(sin(v)./v);
 b=20*log10(H);
 nul=find(b<-40);                              % finds values below 40 dB
 b(nul)=b(nul)-b(nul)-40;                      % sets to -40 if below 40 dB
 subplot(2,3,n)
 plot(theta*180/pi,b)
 title(['kL = ',num2str(kL(n))])
 axis([min(theta*180/pi),max(theta*180/pi),-40,0])
 axis('square')
 set(gca,'xtick',[-90,0,90],'fontsize',10)
 subplot(2,3,n+3)
 [px,py]=pol2cart(theta,b+40);                        % polar plot
 plot(px,py)
 % polar coordinate reference grid
 text([0,48,0],[48,0,-48],['-90';' 0 ';' 90'],...
  'horiz','center','vert','middle','fontsize',10)
 arcx=40*[cos(linspace(pi/2,-pi/2,100)),0];
 arcy=40*[sin(linspace(pi/2,-pi/2,100)),1];
 line(arcx,arcy,'linestyle',':','color',[1 1 1])
 axis equal, axis square, axis off
 view([-90,90])
end
subplot(231)
xlabel('angle (degrees)','fontsize',11)
ylabel('beam pattern (dB)','fontsize',11)
```

G. ALGORITHM c08s06c.m

This algorithm produced Figure 8.7. It illustrates the pressure level at a constant range from a line source.

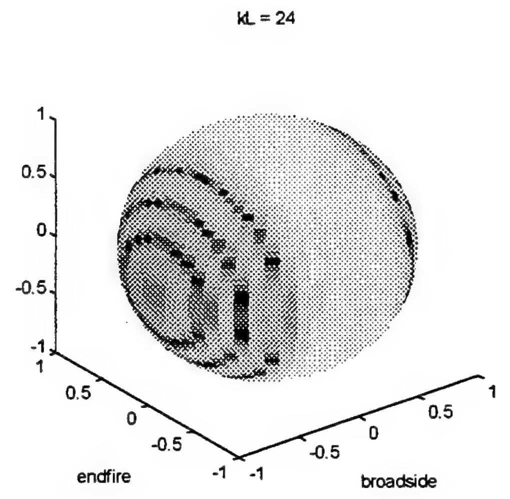

Figure 8.7 Line Source Pressure Level: Spherical Slice

```
% c08s06c.m  line source:  spherical slice
clear, figure(1), clg
kL=24;
n=60;
r=linspace(0,1,n);
```

```
%th=linspace(pi/2,-pi/2,n);                    % azimuth (hemisphere)

th=linspace(0,2*pi,n);                         % azimuth (sphere)

psi=linspace(pi/2,-pi/2,n);                    % elevation

[R,TH,PSI]=meshgrid(r,th,psi);

[X,Y,Z]=sph2cart(TH,PSI,R);

v=abs(kL*cos(sqrt(Y.^2+Z.^2)*pi/2))/2+eps;

H=abs(sin(v)./v);

b=20*log10(H);

nul=find(b<-40);

b(nul)=b(nul)-b(nul)-40;

b=(b+40)./40;

h=slice(R,TH,PSI,b,[1],[],[],n);

for i=1:length(h),

  rn=get(h(i),'xdata');

  az=get(h(i),'ydata');

  el=get(h(i),'zdata');

  [x,y,z]=sph2cart(az,el,rn);

  set(h(i),'xdata',x,'ydata',y,'zdata',z)

end

axis([-1 1 -1 1 -1 1]); axis('square')

shading interp

set(gca,'clim',[.1,1.1])
```

```
title(['kL = ',num2str(kL)])

ylabel('endfire')

xlabel('broadside')

colormap pink
```

H. ALGORITHM c08s06d.m

This algorithm produced Figure 8.8, which is one frame of an animation showing lobe formation of a line source as the frequency is swept.

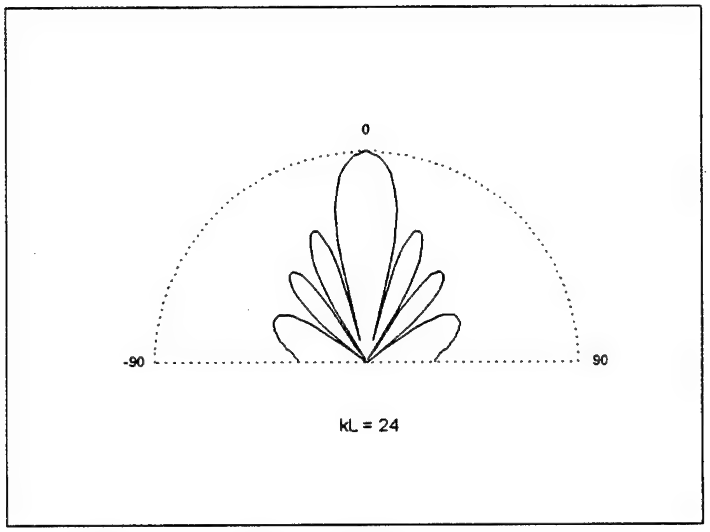

Figure 8.8 Lobe Formation of a Line Source

```
% c08s06d.m  line source:  animation of lobe formation vs freq

clear, figure(1), clf, set(gcf,'backingstore','off')

kL=linspace(eps,24,300);

theta=linspace(-pi/2,pi/2,200);

%%%%%%% polar reference grid

text([0,1.1,0],[1.1,0,-1.1],['-90';' 0 ';' 90'],...

 'horiz','center','vert','middle','fontsize',10)
```

```
arcx=[cos(linspace(pi/2,-pi/2,100)),0];

arcy=[sin(linspace(pi/2,-pi/2,100)),1];

line(arcx,arcy,'linestyle',':','color',[1 1 1])

polarp=line(zeros(size(theta)),zeros(size(theta)),'erasemode','xor');

axis equal, axis square, axis off

view([-90,90])

kLtext=text(-.3,.13,['kL = ',num2str(kL(1))],...

 'erasemode','background','fontsize',12);

%%%%%%% computing beam pattern

for n=1:length(kL);

  v=kL(n)*sin(theta)/2;

  H=abs(sin(v)./v);

  b=20*log10(H);

  nul=find(b<-40);

  b(nul)=b(nul)-b(nul)-40;

  [px(n,:),py(n,:)]=pol2cart(theta,(b+40)./40);

end

%%%%%%% plotting beam pattern

for n=1:length(kL);

  set(polarp,'xdata',px(n,:),'ydata',py(n,:))

  set(kLtext,'string',['kL = ',num2str(kL(n))])

  drawnow
```

```
end
```

I. ALGORITHM c08s08.m

This algorithm produced Figure 8.9. It graphs the pressure amplitude of a circular piston, comparing the exact and far field solutions.

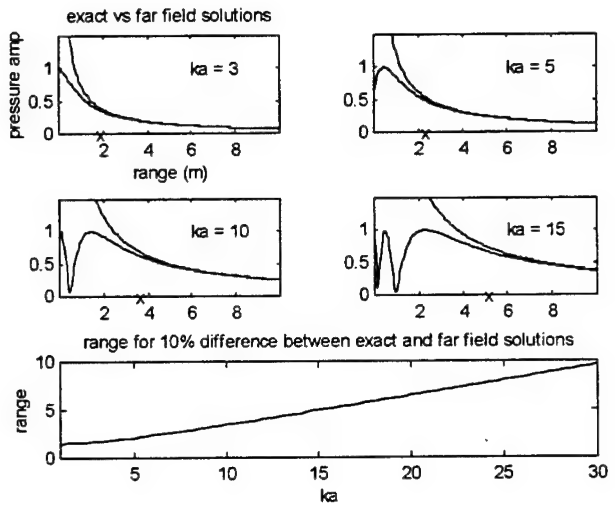

Figure 8.9 Circular Piston Pressure Amplitude

```
% c08s08.m  circular piston:  pressure amplitude
% compares exact solution and far field approximation
% for various values of ka
clear, figure(1), clf
k=[3,5,10,15];                              % wave number
a=1;                                        % radius
r=linspace(.01,10,200);                     % range (on axis)
for n=1:length(k)
  Pe=abs(sin(.5*k(n)*r.*(sqrt(1+(a./r).^2)-1)));    % exact
  Pf=k(n)*a./r./4;                          % far field
  subplot(3,2,n)
  plot(r,Pe,r,Pf)
  axis([min(r),max(r),0,1.5])
  text(6,1,['ka = ',num2str(k(n)*a)],'fontsize',12)
  sep=find(abs((Pf-Pe)./Pf)<.1);
  text(r(sep(1)),0,'x');                              % mark position of 10% difference
end
subplot(321)
xlabel('range (m)')
ylabel('pressure amp')
title('exact vs far field solutions')
sr=[];
k=linspace(1,30,100);
for n=1:length(k)
  Pe=abs(sin(.5*k(n)*r.*(sqrt(1+(a./r).^2)-1)));      % exact
  Pf=k(n)*a./r./4;                                    % far field
  sep=find(abs((Pf-Pe)./Pf)<.1);
  sr=[sr,r(sep(1))];                                  % 10% separaton range
end
subplot(313)
plot(k,sr)
axis([1,max(k),0,10])
```

```
xlabel('ka')

ylabel('range')

title('range for 10% difference between exact and far field solutions')
```

J. ALGORITHM c08s08a.m

This algorithm produced Figure 8.10. It examines the beam pattern of a circular piston.

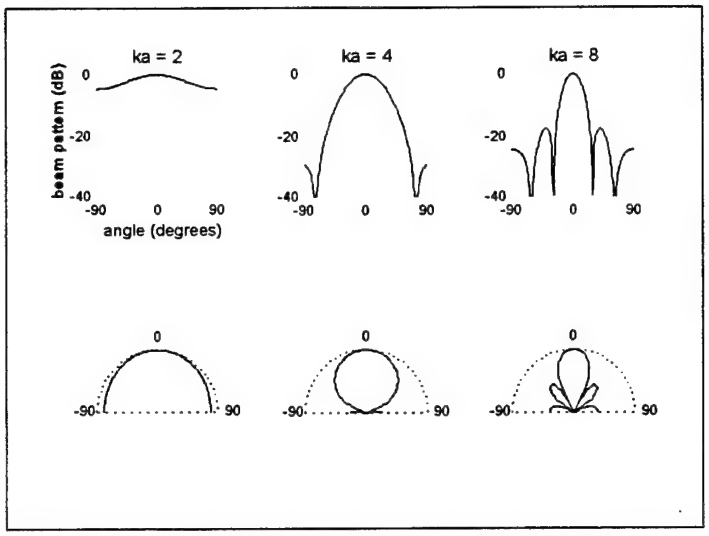

Figure 8.10 Circular Piston Beam Pattern

```
% c08s08b.m  circular piston: beam pattern

clear, figure(1), clf

theta=linspace(-pi/2,pi/2,180);

for n=1:3

  ka=2^n;

  v=abs(ka.*sin(theta));

  J1=besselj(1,v);

  H=abs(2*J1./v);
  b=20*log10(H);
  nul=find(b<-40);                        % finds values below 40 dB
  b(nul)=b(nul)-b(nul)-40;                % sets to -40 if below 40 dB
  subplot(2,3,n)
  plot(theta*180/pi,b)
  title(['ka = ',num2str(ka)])
  axis([min(theta*180/pi),max(theta*180/pi),-40,0])
  axis('square')
  set(gca,'xtick',[-90,0,90],'fontsize',10)
  subplot(2,3,n+3)
  [px,py]=pol2cart(theta,b+40);           % polar plot
  plot(px,py)
  % polar coordinate reference grid
  text([0,48,0],[48,0,-48],['-90';' 0 ';' 90'],...
   'horiz','center','vert','middle','fontsize',10)
  arcx=40*[cos(linspace(pi/2,-pi/2,100)),0];
  arcy=40*[sin(linspace(pi/2,-pi/2,100)),1];
  line(arcx,arcy,'linestyle',':','color',[1 1 1])
  axis equal, axis square, axis off
  view([-90,90])
end
subplot(231)
```

```
xlabel('angle (degrees)','fontsize',11)

ylabel('beam pattern (dB)','fontsize',11)
```

K. ALGORITHM c08s08b.m

This algorithm produced Figure 8.11. It illustrates the pressure level at a constant range from a circular piston.

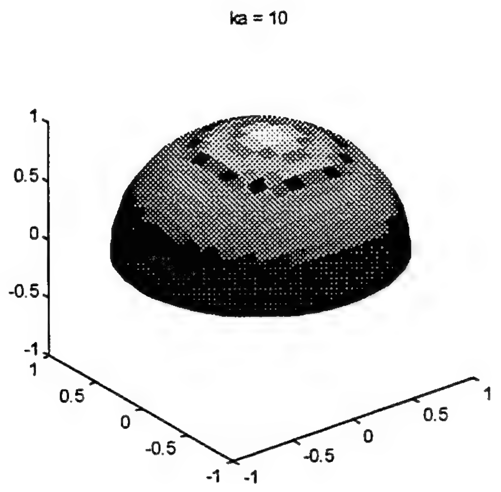

Figure 8.11 Circular Piston: Spherical Slice

```
% c08s08b.m  circular pistion:  spherical slice

clear, figure(1), clg

ka=10;

n=30;

r=linspace(0,1,n);

th=linspace(pi/2,-pi/2,n);                    % azimuth (hemisphere)

psi=linspace(pi/2,-pi/2,n);                   % elevation

[R,TH,PSI]=meshgrid(r,th,psi);

[X,Y,Z]=sph2cart(TH,PSI,R);
```

```
nu=abs(ka*sin(sqrt(Y.^2+Z.^2)*pi/2))+eps;

H=abs(2*besselj(1,nu)./nu);

b=20*log10(H);

nul=find(b<-40);

b(nul)=b(nul)-b(nul)-40;

b=(b+40)./40;

h=slice(R,TH,PSI,b,[1],[],[],n);

for i=1:length(h),

  rn=get(h(i),'xdata');

  az=get(h(i),'ydata');

  el=get(h(i),'zdata');

  [x,y,z]=sph2cart(az,el,rn);

  set(h(i),'xdata',x,'ydata',y,'zdata',z)

end

axis([-1 1 -1 1 -1 1]); axis('square')

shading interp

set(gca,'clim',[.1,1.1])

colormap pink

title(['ka = ',num2str(ka)])

rotate(h,[90,0],90)
```

L. ALGORITHM c08s09.m

This algorithm produced Figure 8.11. It compares the directivity of a circular piston and a line source.

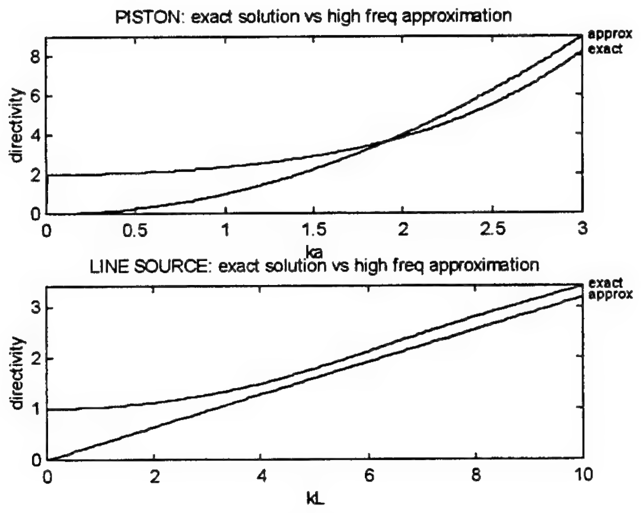

Figure 8.12 Line Source and Circular Piston Directivity

```
% c08s09.m Directivity
% comparison of circular pistion and line source
clear, figure(1), clf
%%%%%%% circular piston
ka=linspace(eps,3,300);
D=ka.^2./(1-besselj(1,2*ka)./ka+eps);          % piston directivity
D1=ka.^2;                                       % high freq approx
subplot(211)
plot(ka,D,ka,D1,'- -')
xlabel('ka'), ylabel('directivity')
```

```
title('PISTON: exact solution vs high freq approximation')

axis([0,max(ka),0,max([D,D1])])

text(max(ka)*1.01,D(length(D)),'exact','fontsize',10)

text(max(ka)*1.01,D1(length(D1)),'approx','fontsize',10)

%%%%%%% line source

kL=linspace(eps,10,200);

dI=0;

dtheta=.0005;

for theta=dtheta:dtheta:pi/2;

  v=.5*kL.*sin(theta);

  H=abs(sin(v)./v);                         % directional factor

  dI=dI+(H.^2).*cos(theta).*dtheta;

end

D=1./dI;                                    % line source

D1=kL/pi;                                   % high freq approx

subplot(212), plot(kL,D,kL,D1,'--')

title('LINE SOURCE: exact solution vs high freq approximation')

xlabel('kL')

ylabel('directivity')

axis([0,max(kL),0,max([D,D1])])

text(max(kL)*1.01,D(length(D)),'exact','fontsize',10)

text(max(kL)*1.01,D1(length(D1)),'approx','fontsize',10)
```

M. ALGORITHM c08s12.m

This algorithm produced Figure 8.13. It compares the radiation impedance of a circular piston and a pulsating sphere.

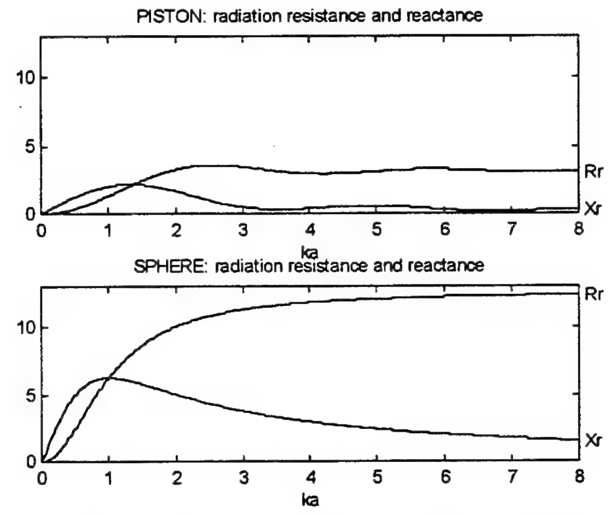

Figure 8.13 Radiation Impedance of Circular Piston and Sphere

```
% c08s12.m  radiation impedance of circular piston and sphere
clear, figure(1), clf
ka=linspace(eps,8,300);
%%%%%%%% circular piston
x=2*ka;                                  % x-axis
R1=1-2*besselj(1,x)./x;
X1=0; neg=-1;
for n=1:2:199
  neg=-neg;
  X1=X1+4/pi*neg*x.^n./prod([[1:2:n].^2,n+2]);
```

```
end

Rrp=pi*R1;                      % piston radiation resistance

Xrp=pi*X1;                      % piston radiation reactance

%%%%%%% sphere

ta=acot(ka);

Zrs=4*pi*cos(ta).*exp(j*ta);    % sphere radiation impedance

Rrs=real(Zrs);                  % sphere radiation resistance

Xrs=imag(Zrs);                  % sphere radiation impedance

%%%%%%% plot

subplot(211)

plot(ka,Rrp,ka,Xrp)

title('PISTON: radiation resistance and reactance')

text(max(ka)*1.01,Rrp(length(Rrp)),'Rr')

text(max(ka)*1.01,Xrp(length(Xrp)),'Xr')

axis([0,max(ka),0,ceil(max([Rrp,Xrp,Rrs,Xrs]))])

xlabel('ka')

subplot(212)

plot(ka,Rrs,ka,Xrs)

title('SPHERE: radiation resistance and reactance')

axis([0,max(ka),0,ceil(max([Rrs,Xrs,Rrs,Xrs]))])

text(max(ka)*1.01,Rrs(length(Rrs)),'Rr')

text(max(ka)*1.01,Xrs(length(Xrs)),'Xr')
```

```
xlabel('ka')
```

N. ALGORITHM c08s13.m

This algorithm produced Figure 8.14. It examines the beam pattern of a linear array without steering.

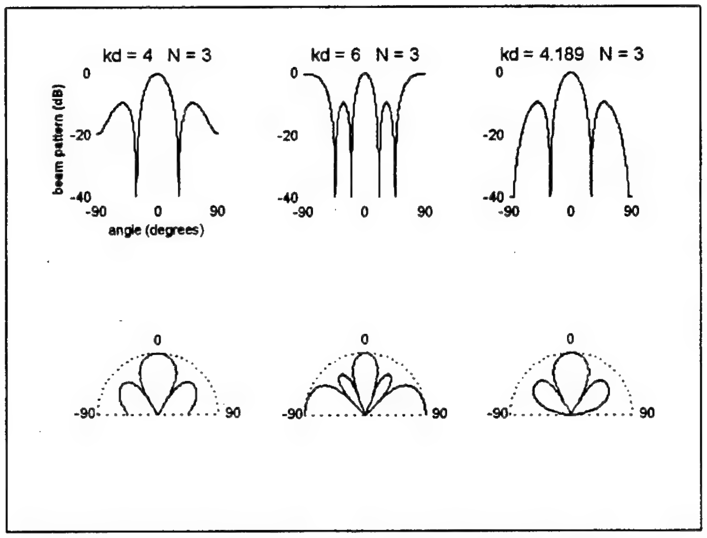

Figure 8.14 Linear Array Beam Pattern (Without Steering)

```
% c08s13.m  linear array beam pattern (without steering)
clear, figure(1), clf
N=3;
kd=[4,6,2*pi*(N-1)/N];
theta = linspace(-pi/2,pi/2,1000);
for n=1:3
  den=sin(N*kd(n)/2.*sin(theta));
  num=sin(kd(n)/2.*sin(theta));
  H=abs((1/N)*den./num);
```

```
DI=(20*log10(H));

nul=find(DI<-40);                                  % find DI < -40 dB

DI(nul)=DI(nul)-DI(nul)-40;                        % if < -40 set to -40 dB

%%%%%%% cartesian coordinates

subplot(2,3,n)

plot(theta*180/pi,DI)

axis([-90,90,-40,0]), axis('square')

title(['kd = ',num2str(kd(n)), '   N = ',num2str(N)],...

 'fontsize',12)

set(gca,'xtick',[-90,0,90],'fontsize',10)

%%%%%%% polar coordinates

DI=DI+40;                                          % set positive for polar plot

[px,py]=pol2cart(theta,DI);

subplot(2,3,n+3)

plot(px,-py)

%%%%%%% polar coordinate reference grid

text([0,48,0],[48,0,-48],['-90';' 0 ';' 90'],...

 'horiz','center','vert','middle','fontsize',10)

arcx=40*[cos(linspace(pi/2,-pi/2,100)),0];

arcy=40*[sin(linspace(pi/2,-pi/2,100)),1];

line(arcx,arcy,'linestyle',':','color',[1 1 1])

axis equal, axis square, axis off
```

```
 view([-90,90])

end

subplot(2,3,1)

xlabel('angle (degrees)')

ylabel('beam pattern (dB)')
```

O. ALGORITHM c08s13a.m

This algorithm produced Figure 8.15. It examines the beam pattern of a linear array with steering.

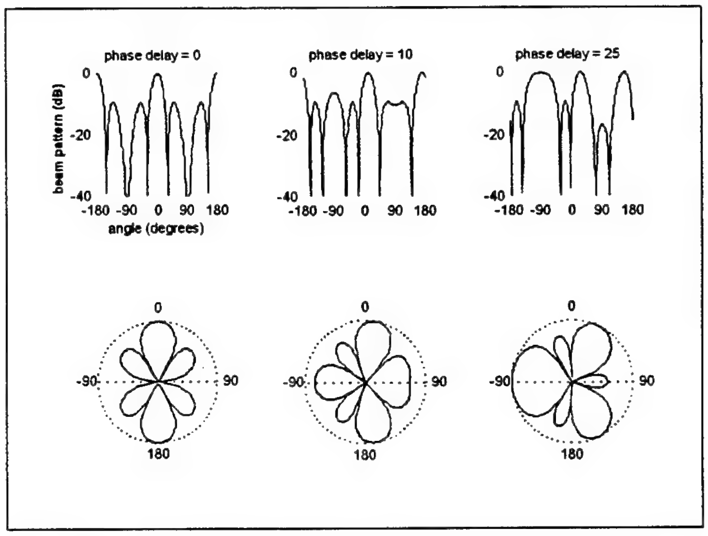

Figure 8.15 Linear Array Beam Pattern (With Steering)

```
% c08s13a.m linear array: beam pattern with steering

% accomplished by linear phase variation across elements

clear, figure(1), clf

N=3;

kd=2*pi*(N-1)/N;
```

```
theta = linspace(-pi,pi,500);
phi=[0,10,25]*pi/180;                          % phase delay (radians)
                                               % pi/2 phase delay = endfire

for n=1:3
 den=sin(N*kd/2.*(sin(theta)-phi(n)));
 num=sin(kd/2.*(sin(theta)-phi(n)));
 H=abs((1/N)*den./num);
 DI=(20*log10(H));
 nul=find(DI<-40);                             % find DI < -40 dB
 DI(nul)=DI(nul)-DI(nul)-40;                         % if < -40 set to -40 dB
 %%%%%%% cartesian coordinates
 subplot(2,3,n)
 plot(theta*180/pi,DI)
 axis([-180,180,-40,0]), axis('square')
 title(['phase delay = ',num2str(phi(n)*180/pi)],...
  'fontsize',10)
 set(gca,'xtick',[-180,-90,0,90,180],'fontsize',10)
 %%%%%%% polar coordinates
 DI=DI+40;                                     % set positive for polar plot
 [px,py]=pol2cart(theta,DI);
 subplot(2,3,n+3)
 plot(px,-py)
 %%%%%%% polar coordinate reference grid
 text([0,48,0,-48],[48,0,-48,0],['-90';' 0 ';' 90';'180'],...
  'horiz','center','vert','middle','fontsize',10)
 arcx=40*[cos(linspace(-pi/2,3*pi/2,300)),0];
 arcy=40*[sin(linspace(-pi/2,3*pi/2,300)),1];
 line(arcx,arcy,'linestyle',':','color',[1 1 1])
 axis equal, axis square, axis off
 view([-90,90])
end
subplot(2,3,1)
xlabel('angle (degrees)')
ylabel('beam pattern (dB)')
```

P. ALGORITHM c08s13b.m

This algorithm produced Figure 8.16. It examines the beam pattern of a linear array with amplitude shading.

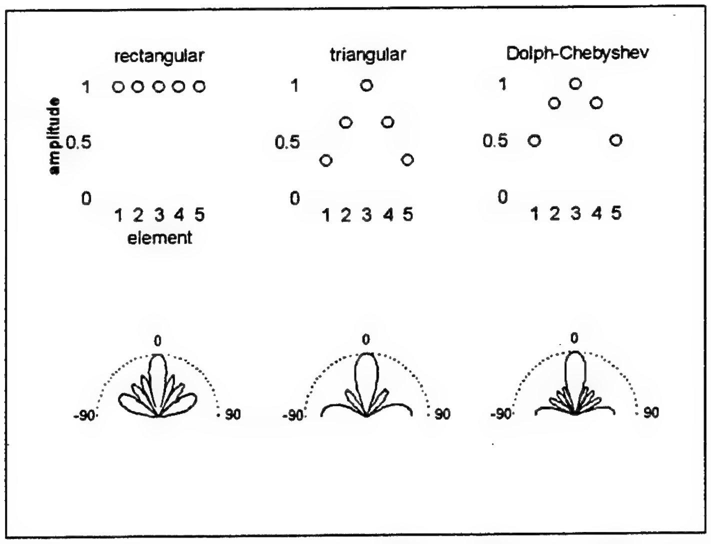

Figure 8.16 Linear Array Beam Pattern (With Amplitude Shading)

```
% c08s13b.m  linear array:  amplitude shading
clear, figure(1), clf
N=5;                              % number of elements
d=1;                              % element spacing
k=2*pi*(N-1)/N/d;                 % wave number
r=100*N;                          % range (far field)
theta=linspace(-pi/2,pi/2,300);
y=r*cos(theta);
x=r*sin(theta);
xn=[0:N-1]*d-(N-1)*d/2;           % element positions
w(:,1)=ones(N,1);                 % rectangular
w(:,2)=triang(N);                 % triangular
```

```
w(:,3)=chebwin(N,20);                 % Dolph-Chebyshev

dP=zeros(length(theta),3);

%%%%%%% summation of element pressures

for n=1:length(xn)

 r1=sqrt((x-xn(n)).^2+y.^2);

 for m=1:3

   dP(:,m)=(exp(j*k*r1)./r1*w(n,m))'+dP(:,m);

 end

end

[mdP,mw]=meshgrid(1./max(abs(dP)),1:length(theta));

H=abs(dP).*mdP;

plot(theta,H)

D=20*log10(H);                        % beam pattern (3 column matrix)

nul=find(D<-40);                      % find D < -40 dB

D(nul)=D(nul)-D(nul)-40;              % trim to -40 dB

D=D+40;                               % set positive for polar plot

for n=1:3

  %%%%%%% amplitude weight plot

  subplot(2,3,n)

  plot(1:N,w(:,n),'o')

  axis([0,N+1,0,1.1]), axis square

  if n==1, title('rectangular')
 elseif n==2, title('triangular')
 elseif n==3, title('Dolph-Chebyshev')
 end
 set(gca,'xtick',1:N)
 %%%%%%% polar coordinate plot
 [px,py]=pol2cart(theta',D(:,n));
 subplot(2,3,n+3)
 plot(px,-py)
 %%%%%%% polar coordinate reference grid
 text([0,48,0],[48,0,-48],['-90';' 0 ';' 90'],...
  'horiz','center','vert','middle','fontsize',10)
 arcx=40*[cos(linspace(-pi/2,pi/2,300)),0];
 arcy=40*[sin(linspace(-pi/2,pi/2,300)),1];
 line(arcx,arcy,'linestyle',':','color',[1 1 1])
 axis equal, axis square, axis off
 view([-90,90])
end
subplot(231)
xlabel('element')
ylabel('amplitude')
```

LIST OF REFERENCES

KFCS Kinsler L. E., Frey A. R., Coppens A. B., Sanders J. V.., "Fundamentals of Acoustics", Third Edition, John Wiley & Sons, Inc., Monterey, CA, 1980

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Road., Ste 0944 Ft. Belvoir, Virginia 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, California 93943-5101	2
3.	Undersea Warfare, Code 37 Naval Postgraduate School Monterey, California 93943-5002	1
4.	Dr. James V. Sanders, Code PH/Sd Department of Physics Naval Postgraduate School Monterey, California 93943-5002	2
5.	Dr. Anthony A. Atchley PH/Ay Department of Physics Naval Postgraduate School Monterey, California 93943-5002	1
6.	Dr. Al Coppens P.O. Box 335 Black Mountain, North Carolina 28711-0335	1
7.	Dr. Robert M. Keolian, Code PH/Kn Department of Physics Naval Postgraduate School Monterey, California 93943-5002	1
8.	LCDR Thomas A. Green 5542 W. Cambridge Ave. Phoenix, Arizona 85035	2